NM ZUSAMMENFASSUNG

SIMONE GUGGIARI
ETH ZÜRICH 2015
LEGI # 14-932-867

0. INTRODUCTION

Computing with Matrices $\underline{\underline{M}}$ and vectors $\vec{v}$

$v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \in \mathbb{K}^n$, $\mathbb{K} = \mathbb{R}, \mathbb{C}$ $\quad A = \begin{bmatrix} a_{11} & \cdots & a_{1,m} \\ a_{n,1} & & a_{n,m} \end{bmatrix} \in \mathbb{K}^{n,m}$

$(v)_i := v_i$ $\quad (A)_{i,j} = a_{ij}$ $\quad$ sub vectors/mat blocks $(v)_{k:l} = \begin{bmatrix} v_k \\ \vdots \\ v_l \end{bmatrix}$ $\quad (A)_{k:l,c:s} \in \mathbb{K}^{l-k+1,\, s-c+1}$

Software & Libraries

MATLAB - IDE for numerical computation - everything is a matrix $\quad v = [1,2;3,4] \to \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$
- loop over column of matrix
- index from 1

- **EIGEN** - header-only C++ library for numerical algebra
- index from 0

matrix storage format → stored in linear form, es $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ → row major (C-arrays, ...) [1 2 3 4 5 6]
→ col major (MATLAB, EIGEN, ...) [1 4 2 5 3 6]

mapping (row major) $A(i,j) = A[n(i-1)+j]$
↳ impact on runtime (es how blocks are stored in cache) ↳ faster, because contiguous access to memory

1. COMPUTING WITH MATRICES & VECTORS

Basic lin. alg operations $+ \quad - \quad * \quad /$

- **BLAS** (basic linalg subprograms) = highly optimized low level linalg functions $\quad O(n) \quad \forall\forall$

Wl 1: vector op (scalar products, vector norms)
- dot product $\rho = x^T y$ $\quad$ x DOT (N, x, INCX, Y, INCY)
↳ {S, D} ↳ vector ↳ step to traverse x
- vector op $y = \alpha x + y$ $\quad$ x AXPY (N, ALPHA, X, INCX, Y, INCY)
↳ {S, D, C, Z}

Wl 2: vector/matrix op $\quad O(mn) \quad \forall\Pi$
- matrix/vector multiplication $y = \alpha Ax + \beta y$
x GEMV (TRANS, M, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
↳ size ↳ matrix ↳ leading dimension of A (n) ↳ stride (linear)

Wl 3: matrix op (matrix addition/mult) $\quad O(mnk) \quad \Pi\Pi$
- matrix matrix mult $C = \alpha AB + \beta C$
x GEMM (TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, X, B, LDB, BETA, C, LDC)

scaling: row scaling: mult diag matrix from LEFT $\quad DA = \begin{pmatrix} d_1 (A)_{1,:} \\ \vdots \\ d_n (A)_{n,:} \end{pmatrix}$
col RIGHT $\quad AD = \begin{pmatrix} d_1 (A)_{:,1}, \dots, d_m (A)_{:,m} \end{pmatrix}$

computational effort required by num. code = $\frac{\text{number of elementary op}}{\text{(modern PC - num. hierarchy, internal pipelining, vectorization, ...)}}$ $(+ - / * \sqrt{\ })$
↳ it's only loosely related to overall execution time

asymptotic complexity of alg characterizes worst-case dependence of comp. effort depending on problem size parameter when $\to \infty$
$O(n^p)$: comp. eff $\le C \cdot n^p$ (strict bound: assume $\not\le Cn^q \ \forall q < p$)
complexity $\not\Rightarrow$ runtime

cost of basic op

op	description	#mult/div	#add/sub	opt ?
dot prod	$x \in \mathbb{R}^n, y \in \mathbb{R}^n \to x^H y$	n	$n-1$	$O(n)$ ✓
tensor prod	$\mathbb{R}^m \to xy^H$	nm	0	$O(mn)$ ✓
matrix prod	$A \in \mathbb{R}^{m,n}, B \in \mathbb{R}^{n,k} \to AB$	mnk	$mk(n-1)$	$O(mnk)$ ✗ ↳ strass alg $O(n^{2,3})$, D&C

TRICKS

- **exploit associativity** es $y = (ab^T)x$ $\quad O(n^2)$ $\quad$ vs $\quad y = a(b^T x)$ $\quad O(n)$

- **hidden summation** es $y = \text{trio}(A * B') * x$ $\quad$ $y = \text{zeros}(n,1)$; for $l = 1:p$ $y = y + A(:,l) .* \text{cumsum}(B(1,l) * x, \text{'reverse'})$; $\quad O(np)$
$A, B \in \mathbb{R}^{n,p}, p \ll n$
you can be more efficient by reusing intermediate results
cumsum → cumulative sum

testing $x == 0$ for x float is dangerous → test $\|x\| \approx 0$ in relative sense

Kronecker product $A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots \\ a_{21}B & \ddots & \\ \vdots & & a_{mn}B \end{bmatrix}$

MACHINE ARITHMETIC

- es bss of orthogonality (gram-schmidt)

Hilbert $\underline{\underline{M}}$: $(A)_{ij} = \frac{1}{i+j-1} \to \begin{pmatrix} 1 & 1/2 & \cdots \\ 1/2 & 1/3 & \\ \vdots & & \ddots \end{pmatrix}$

QR-decomposition $A \in \mathbb{R}^{n,k} \to Q \times R \in \mathbb{R}^{n,k} \times \mathbb{R}^{k,k}$ $\quad$ economic $\quad \square = \square\square$
$QQ = I$ upp. tr. $\in \mathbb{R}^{n,n} \times \mathbb{R}^{n,k}$ $\quad$ full $\quad \square = \square\!\!\square$

MATLAB: Gram-Sch $\neq$ QR (bad) (good) → due to cancellation & roundoff

- **Machine numbers** (= floats)
PC can handle finite # of numbers $\quad \mathbb{M} \subset \mathbb{R}$ finite, discrete subset

$\text{op}: \mathbb{M} \times \mathbb{M} \to \mathbb{R}$ $\quad \not\to \mathbb{M}$ (not a closed set) $\quad \widetilde{\text{op}} = \text{rd} \circ \text{op} = \text{rd}(\text{op})$
replace with ↓ $\quad$ correct rounding: $\text{rd}: \begin{cases} \mathbb{R} \to \mathbb{M} \\ x \to \max \operatorname{argmin} \{|x - \tilde{x}| : \tilde{x} \in \mathbb{M}\} \end{cases}$ (round up, guello de numero)
$\widehat{\text{op}}: \mathbb{M} \times \mathbb{M} \to \mathbb{M}$

$x_{max} = \max |\mathbb{M}| = (1 - B^{-m}) \cdot B^{e_{max}}$ $\quad$ single prec: $m = 24$ $\quad e \in \{-126, 127\}$
$x_{min} = B^{-1} \cdot B^{e_{min}}$ $\quad$ double $\quad m = 53$ $\quad e = 11$, $\quad -1022, 1023$
sign bit included

special cases

	$m = 0$	$m \neq 0$
$e = 0$	0	denorm. number → $(-1)^s \cdot 0.m \cdot 2^{-e_{min}}$
$e = e_{max}$	$\pm \infty$	NaN

roundoff errors

$\tilde{+}, \tilde{\times}$ not associative! (due to roundoff)

$\text{eps} = \max_{x \in \mathbb{R} \setminus \{0\}} \frac{|x - \text{rd}(x)|}{|x|} = \frac{1}{2} B^{1-m}$ $\quad$ maximal relative roundoff error
machine precision

small, positive
smallest # for which $1 + \varepsilon \neq 1$

axiom of roundoff analysis

$\forall$ element op $\{+, -, *, /\}$ and functions $\{\exp, \sin, \cos, \log, \dots\}$ $\quad$ result guaranteed not to exceed this bound

$\Rightarrow x \tilde{*} y = (x * y)(1 + \delta)$, $\quad \tilde{f}(x) = f(x)(1 + \delta)$, $\quad |\delta| < \text{EPS}$

cancellation

these tiny error can amplify / accumulate

- roots of quadratic polynomial $\quad x^2 + \alpha x + \beta = 0$

stable discriminant formula
1) compute large root without cancellation (z
2) $z_2 = \beta / z_1$

$[\ ] - [\ ] =$ subtraction of almost identical numbers (amplification of relative errors)

- approximation of derivative $\quad f'(x) \approx \frac{f(x+h) - f(x)}{h}$, $\quad h \ll 1$

roundoff error analysis (use axiom)

$\Delta f = \frac{e^{x+h}(1+\varepsilon_1) - e^x(1+\varepsilon_2)}{h} = e^x\left(\frac{e^h - 1}{h} + \frac{\varepsilon_1 e^h - \varepsilon_2}{h}\right) = e^x\left(1 + \delta^*(h)\right)$
↳ cancellation error $\quad$ ↳ δ^* minimal for $h \approx \sqrt{\text{eps}}$

avoid cancellation

- log identities $\quad \int_0^x \sin t\, dt = \underbrace{1 - \cos x}_{\text{canc. if } x \ll 1} = 2\sin^2(x/2)$ ↳ no cancel

- e^x by truncated exp series $\quad e^x \approx \sum_{k=0}^{n} \frac{1}{k!} x^k$ $\quad \to$ if negative, adding terms with alternating signs → use $e^{-x} = 1/e^x$

taylor approx

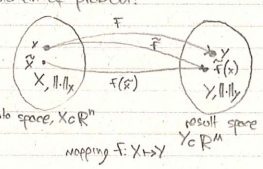
$T_n f(x;a) = \sum_{n=0}^{n} \frac{f^{(n)}(a)}{n!}(x-a)^n$ $\quad$ es $\frac{1}{a}(e^a - 1) = \sum_{k=0}^{n} \frac{1}{(k+1)!} a^k + \frac{1}{(n+1)!} e^{\xi} a^{n+1}$
$=: S_n(a)$ $\quad$ remainder

rel. error $= \left|\frac{S_n(a) - I(a)}{I(a)}\right| \le \text{tol}$

determine n to have desired precision ↳ $\frac{1}{(n+1)!} e^a a^{n+1} \le \text{tol}$ (n from a priori)

Numerical stability

notation of problem:



data space, $X \subset \mathbb{R}^n$ $\quad$ result space $Y \subset \mathbb{R}^m$
mapping $f: X \to Y$

stable alg: alg $\tilde{f}$ for solving $f: X \to Y$ is numerically stable if $\forall x \in X$, $\tilde{f}(x)$ (affected by roundoff) is exact result of slightly perturbed data

$\exists C \ge 1: \forall x \in X \ \exists \tilde{x} \in X: \|x - \tilde{x}\|_X \le C \cdot \text{eps} \cdot \|x\|_X$ $\quad \tilde{f}(x) = f(\tilde{x})$

→ impact of roundoff during execution is not much more than effect of rounding data

- es stability matrix×vector

function $y = \text{mult_mv}(A, x)$ $\quad$ stable?

$F: \begin{cases} X := \mathbb{K}^{m,n} \times \mathbb{K}^n \to \mathbb{K}^m =: Y \\ (A, x) \to Ax \end{cases}$

given $\tilde{y}$, when is there $\tilde{A}$ such that $\tilde{A}x = \tilde{y}$
$\|A - \tilde{A}\|_2 \le C \cdot n \cdot \|A\|_2$
→ poss. choice: $\tilde{A} = A + zx^T$, $z = \frac{\tilde{y} - Ax}{\|x\|^2}$

$\tilde{A}x = Ax + z\|x\|^2 = Ax + \tilde{y} - Ax = \tilde{y}$

$\|A - \tilde{A}\|_2 = \|zx^T\|_2 = \sup_{w \neq 0} \frac{\|zx^T w\|_2}{\|w\|_2} = \|z\|_2 \|x\|_2 = \|\tilde{y} - Ax\| / \|x\|_2$

$\frac{\|A - \tilde{A}\|_2}{\|A\|_2} \le C \cdot n \cdot \text{eps}$ $\quad$ if $\|\tilde{y} - Ax\|_2 \frac{1}{\|A\|_2 \|x\|_2} \le Cn\,\text{eps}$

rule: If problem is **sensitive** ($\|f(x) - f(\tilde{x})\| \gg 1$ though $x \approx \tilde{x}$)
⇒ easy to find stable alg!

sensitivity → problem
stability → (depends on) alg

ill conditioned: small input variation produce big output errors

Matrix norms

$\|M\|_2 := \sup_{x \in \mathbb{R}^n \setminus \{0\}} \frac{\|Mx\|_2}{\|x\|_2}$

$\|M\|_1 = \max$ abs col sum
$\|M\|_\infty =$ row

submultiplicative: $\|AB\| \le \|A\| \|B\|$
⚠ mult. with unitary matrix preserves norm!

DIRECT METHODS FOR LINEAR EQ. SYSTEMS

⇒ given A, b find x such that $Ax = b$

$F: (A, b) \mapsto A^{-1} b$

stability: $Ax = b \Leftrightarrow A\tilde{x} = \tilde{b}$ $\quad \frac{\|b - \tilde{b}\|}{\|b\|} = \frac{\|b - A\tilde{x}\|}{\|b\|} = \frac{\|r\|}{\|b\|}$, $r := b - A\tilde{x}$ residual
↳ stable if $\|r\| \le C \cdot n \cdot \text{eps} \cdot \|b\|$

elimination solvers for LSE

cost: $O(n^3)$ → cheaper for special matrices
- diagonal $O(n)$
- unitary $O(n^2)$ $\quad x = A^H b$
- triangular $O(n^2)$ - forward/back

MATLAB: $x = A \backslash b$ (automatically selects best option)
EIGEN: $X = A.\langle \text{dec} \rangle.\text{solve}(B)$
↳ method of solving

elimination solve = setup phase $O(n^3)$ + elimination phase $O(n^2)$

conditioning of LSE

$\text{cond}(A) = \|A\| \|A^{-1}\|$ expresses how much small perturbations can lead to big rel. errors (how much col/rows are almost lin. dep)

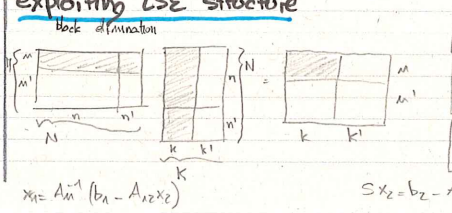
$\frac{\|x - \tilde{x}\|}{x} \le \frac{\|A\| \cdot \|A^{-1}\|}{1 - \|A^{-1}\| \cdot \|A\| \frac{\|\Delta A\|}{\|A\|}} \left(\frac{\|\Delta b\|}{\|b\|} + \frac{\|\Delta A\|}{\|A\|}\right)$

perturbation lemma

$B \in \mathbb{R}^{n,n}$, $\|B\| < 1 \Rightarrow I + B$ regular $\wedge$ $\|(I+B)^{-1}\| \le \frac{1}{1 - \|B\|}$

exploiting LSE structure

block elimination



$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$

$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$

$\begin{bmatrix} \cdot & 0 \\ \cdot & \cdot \end{bmatrix} = \begin{bmatrix} I & 0 \\ A_{21}A_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} \\ 0 & S \end{bmatrix}$ $\quad A_{22} - A_{21}A_{11}^{-1}A_{12}$
↳ schur complement

$x_1 = A_{11}^{-1}(b_1 - A_{12}x_2)$ $\quad S x_2 = b_2 - A_{21}A_{11}^{-1} b_1$

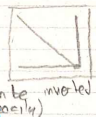
Solving LSE with LU

$Ax=b$, $A=LU \Rightarrow LUx=b$. 1) $Ly=b$ (forward) 2) $Ux=y$ (backward)

arrow system matrix

$b,c \in \mathbb{R}^n$, $D \in \mathbb{R}^{n,n}$ diag, $\alpha \in \mathbb{R}$

$$\begin{bmatrix} D & c \\ b^T & \alpha \end{bmatrix}\begin{bmatrix} x_i \\ \xi \end{bmatrix} = \begin{bmatrix} y_i \\ \eta \end{bmatrix}$$



start: Back elimination (instead of inverting)

$O(n)$ vs $O(n^3)$

$$\Rightarrow (\alpha - b^T D^{-1} c)\xi = \eta - b^T D^{-1} y_i \quad \rightarrow D^{-1}c : z = c./d$$

↳ first it

$$x_i = D^{-1}(y_i - c\xi)$$

How to tell polyn. complexity from runtime data (n_i, t_i)

if $t_i \cong C \cdot n_i^p \Rightarrow \log t_i \cong \log C + p \log(n_i)$ → straight line with slope p in log-log plot

Low rank modification of LSE

assume $Ax=b$ has already been solved

sought: $\tilde{x}$: $\tilde{A}\tilde{x}=b$, where $\tilde{A}$ comes from A, with 1 entry changed

$$(\tilde{A})_{ij} = \begin{cases} (A)_{ij} & \text{if } (i,j) \neq (i^*,j^*) \\ z + (A)_{ij} & \text{else} \end{cases} \qquad \tilde{A} = A + z\, e_{i^*} e_{j^*}^T \rightarrow \text{0-matrix with single "1" entry}$$

rank 1 modification

$$\tilde{A} = A + UV^T \qquad \begin{bmatrix} A & U \\ V^T & -1 \end{bmatrix}\begin{bmatrix} \tilde{x} \\ \xi \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix} \rightarrow \begin{matrix} \xi = V^T\tilde{x} \\ (A + UV^T)\tilde{x} = b \end{matrix}$$

$$A\tilde{x} = b - U\frac{V^T A^{-1} b}{1 + V^T A^{-1} U} \qquad x = A^{-1}b - \frac{A^{-1}U\,(V^H (A^{-1} b))}{1 + V^H (A^{-1} U)}$$

Sherman-Morrison-Woodbury formula

$$(A + UV^H)^{-1} = A^{-1} - A^{-1}U\,(I + V^H A^{-1} U)^{-1} V^H A^{-1}$$

SPARSE LINEAR SYSTEM (most entries of matrix = 0)

$A \in \mathbb{K}^{n,m}$ is sparse $\Leftrightarrow \text{nnz}(A) := \#\{(i,j) \in \{1,\dots,n\} \times \{1,\dots,m\} : a_{ij} \neq 0\} \ll nm$ $\quad = O(n+m)$

sparse matrix storage format

memory $\cong O(\text{nnz}(A))$

cost (Matrix × Vector) $\cong O(\text{nnz}(A))$

triplet format (COO) – coordinates $:= \text{List}\{(i_k, j_k, a_k)\}_{k=1}^{N}$

repetition of index possible → add them (row/col index ↳ entry)

- **CRS** (compressed row format) $A \in \mathbb{K}^{n,m}$

vector<scalar_t> val (unit double) — size nnz(A)

vector<size_t> col_ind — size nnz(A)

vector<size_t> row_ptr — size n+1 → begin of i^{th} row (pos of first nonzero entry)

↳ sentinel value := nnz(A)+1

$$\text{val}[k] = a_{ij} \Leftrightarrow \begin{cases} \text{col_ind}[k] = j, \\ \text{row_ptr}[i] \le k < \text{row_ptr}[i+1] \end{cases} \qquad 1 \le k \le \text{nnz}(A)$$

ex $\begin{bmatrix} 10 & 0 & 0 & 0 \\ 3 & 9 & 0 & 0 \\ 0 & 7 & 8 & 7 \\ 3 & 0 & 8 & 7 \end{bmatrix}$

10 3 9 7 8 7 3 2 7

1 1 2 2 3 4 1 3 4

1 2 4 7 10

CCS = CRS for A^T

sparse in MATLAB → see dedicated fz

→ to be efficient, allocate needed space at beginning! otherwise too much copy & malloc (keep making it bigger)

sparse in EIGEN → see API's

→ remark: reserve() method for sparse → preallocate space

→ most efficient initialization: coeffRef with reserved space

direct solutions of sparse lin. systems

(system matrix in sparse format) – use special sparse elimination algorithms (dedicated to that)

MATLAB: simply use \

SMOOTHING OF PLANAR TRIANGULATION

Def: planar triangulation: A planar triang (mesh) $\mathcal{M}$ consists of a set $\mathcal{N}$ of $N \in \mathbb{N}$ distinct points $\in \mathbb{R}^2$ and a set $\mathcal{T}$ of triangles with vertices in $\mathcal{N}$, such that following holds:

1. Interiors of triangles are mutually disjoint
2. $\forall$ 2 closed distinct triangles, their intersection (or) • is empty • is one vertex • is a common edge
3. it is fully covered

distorted → move interior node to smooth

smoothed triangulation

$S(i) := \{j \in \{1,\dots,N\} : \text{nodes } i,j \text{ connected by edge}\}$, $p^i :=$ pos of node i

triangulation is called smoothed, if

$$p^i = \frac{1}{\#S(i)} \sum_{j \in S(i)} p^j \qquad \forall\, i \in \{1,\dots,N\} \setminus \Gamma$$

(every node is in center of gravity) ↳ border

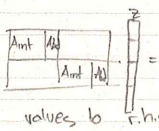
$z \in \mathbb{R}^{2N}$ (vector of node coord) $\quad z_j = \begin{cases} p_x^j, & 1 \le j \le N \\ p_y^{j-N}, & N+1 \le j \le 2N \end{cases}$

$$C = \begin{bmatrix} A & 0 \\ 0 & A \end{bmatrix}, \quad A \in \mathbb{R}^{N,N} \qquad (A)_{ij} = \begin{cases} -\#S(i) & \text{if } i=j \\ 1 & \text{if } j \in S(i) \\ 0 & \text{else} \end{cases}$$

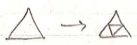
is sparse Laplacian combinatorial graph

position of boundary nodes are known

$$z^T = \begin{bmatrix} z_1^{int} \\ z_1^{bd} \\ z_2^{int} \\ z_2^{bd} \end{bmatrix} := [z_1, \dots, z_n, z_{n+1}, \dots, z_N, z_{N+1}, \dots, z_{n+N}, z_{n+N+1}, \dots, z_{2N}]^T$$

$Cz = 0 \Leftrightarrow$ 

move known values to r.h.s $\Rightarrow \begin{bmatrix} A_{int} & 0 \\ 0 & A_{int} \end{bmatrix}\begin{bmatrix} z_1^{int} \\ z_2^{int} \end{bmatrix} = \begin{bmatrix} -A_{bd} z_1^{bd} \\ -A_{bd} z_2^{bd} \end{bmatrix}$

regular refinement  →

see spy-plot

#primes 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, ... (2)

partial sum formula $\sum_{i=0}^{n} q^i = \frac{q^{n+1}-1}{q-1}$ **infinite sum formula** $\sum_{i=0}^{\infty} q^i = \frac{1}{1-q}$ if $|q|<1$

$$\sum_{k=j+1}^{\infty} \binom{j}{k} = \left(\frac{1}{2}\right)^{j+1} \sum_{k=0}^{\infty} \binom{j}{k}$$

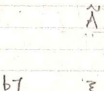
exponential series $\sum_{k=0}^{\infty} \frac{x^k}{k!} = e^x$ **exponential limit** $e^x = \lim_{n\to\infty} \left(1 + \frac{x}{n}\right)^n$

binomial formula $(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^{n-k} b^k$

2. ITERATIVE METHODS FOR NON-LINEAR EQ. SYST

general problem: given $F: D \subset \mathbb{R}^n \rightarrow \mathbb{R}^n$, find $x \in \mathbb{R}^n$ such that $F(x) = 0$ (F might be black box)

case study: nodal analysis of electric circuit



$I_{kj} = -I_{jk}$ current node $k \rightarrow j$

U_k = nodal potential

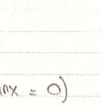
Kirchoff laws: $\sum_{j \in S(k)} I_{kj} = 0$

$I_{kj} = \alpha\,(U_k - U_j)$

$\alpha = \begin{cases} R^{-1} & \text{resistor} \\ i\omega C & \text{capacitor} \\ (-i/\omega L) & \text{coil} \end{cases}$ ω = angular frequency

$I(t) = \text{Re}(I e^{i\omega t}) = \text{Re}(I)\cos(\omega t) - \text{Im}(I)\sin(\omega t)$

nodal analysis: 1. $\forall$ node (unknown potential) 2. replace all currents

Schmitt trigger (NPN) transistor 

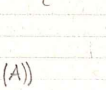
(here we look for the U)

$I_C = \dots = I_C(U_{BE}, U_{BC})$

$I_B = \dots = I_B(\dots)$

$I_E = \dots = I_E(\dots)$

ITERATIVE METHODS: approximately solving non-lin eq $F(x)=0$ generates an arbitrary long sequence $(x^{(k)})_k$ of approximate solutions



iteration error $e^{(k)} = x^{(k)} - x^*, \quad \varepsilon_k = \|x^{(k)} - x^*\|$

stationary m-point iteration $x^{(k+1)} = \Phi_F(x^k, \dots, x^{(k-m+1)})$ ↳ iterative fct

initial guess $x^{(0)}, x^{(1)}, \dots, x^{(m-1)} \in \mathbb{R}^n$ ↳ it matters much!

issues: • well defined (= unambiguous)? • does it converge? • if yes, to a solution? (consistency) • how fast?

local & global convergence: local conv. to x^*, if $\exists$ neighborhood $U \subset D$ of x^*, s.t. $x^{(0)}, \dots, x^{(m-1)} \in U \Rightarrow x^{(k)}$ well defined $\wedge \lim_{k\to\infty} x^{(k)} = x^*$

global if $U = D$

speed of convergence

linear: $x^{(k)}$ conv. linear to $x^* \Leftrightarrow \exists L < 1 : \|x^{(k+1)} - x^*\| \le L \|x^{(k)} - x^*\| \quad \forall k \in \mathbb{N}_0$ (smallest possible L → rate of conv)

how to test: L.C: assume $\varepsilon_{k+1} \cong L\varepsilon^k \cong L^2 \varepsilon^{k-1} \cong \dots \cong L^{k+1}\varepsilon_0$

$\Rightarrow \log\varepsilon_k \cong k \log L + \log\varepsilon_0 \Rightarrow (k, \log\varepsilon_k)$ on a straight line

from tabulated values, check if $\frac{\varepsilon_k}{\varepsilon_{k-1}} \cong L$

order of conv $x^{(k)}$ in $\mathbb{R}^n$ conv with order p to x^* if $\exists C > 0 : \|x^{(k+1)} - x^*\| \le C \|x^{(k)} - x^*\|^p \quad \forall k \in \mathbb{N}_0$ → algebraic ($C<1$ if $p=1$)

identify: $\{\varepsilon_{k+1} \cong C \cdot \varepsilon_k^p\} \Rightarrow \log\varepsilon_{k+1} \cong \log C + p \log\varepsilon_k$

$$p = \frac{\log\varepsilon_{k+1} - \log\varepsilon_k}{\log\varepsilon_k - \log\varepsilon_{k-1}}$$

↳ if constant → algebraic conv!

ex: $\sqrt{\ }$ iteration $x^{(k+1)} = \frac{1}{2}\left(x^{(k)} + \frac{a}{x^{(k)}}\right)$, $a>0$: $x^{(k)} \rightarrow \sqrt{a}$ (it's a special case of Newton's)

quadratic conv ($p=2$) → # of correct digits doubles

$x^{(k)} = x^*(1 + \varepsilon_k)$ ↳ rel. error: $\varepsilon_k \le 10^{-\ell} \Leftrightarrow x^{(k)}$ has ℓ correct digits!

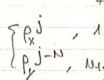
AGM: (arithm-geom mean equality) $\sqrt{ab} \le \frac{1}{2}(a+b)$

TERMINATION

ideal: STOP if $x^{(k)}$ is good enough $\|x^{(k)} - x^*\| \le \tau_{Abs}$ (abs tol) $\le \tau_{rel} \cdot \|x^*\|$ (rel tol)

not practical → x^* unknown

residual based termination STOP if $\|F(x^{(k)})\| \le \tau$ → also, waste of resources



$\|F(x^{(k)})\|$ small $\not\Rightarrow$ $|x - x^*|$ small if F flat

$\Rightarrow$ if f steep

correction based termination STOP if $\|x^{(k+1)} - x^{(k)}\| \le \begin{cases} \tau_{Abs} \\ \tau_{rel} \cdot \|x^{(k+1)}\| \end{cases}$

generally no guarantees, except lin. conv with known rate $L<1$

$\|x^{(k+1)} - x^*\| \le \frac{L}{1-L} \|x^{(k)} - x^{(k+1)}\|$ ↳ reliable upper bound to replace in termination criteria overest. $L \Rightarrow$ still reliable

prior term: $\|x^{(k)} - x^*\| \le \frac{L^k}{1-L} \|x^{(1)} - x^{(0)}\|$

IM based termination iterate until solution becomes stationary. Very inefficient, could be ∞-loop

prior term. stop after fixed # of steps, possibly depending on $x^{(0)}$. Nice but rarely possible

posterior term. use already computed iterates to decide when to stop

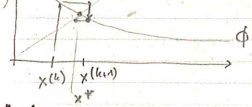
FIXED POINT ITERATIONS

$x^{(k+1)} = \phi_f(x^{(k)})$ = 1-point method, with iterative fn $\phi_f : M \subset \mathbb{R}^n \to \mathbb{R}^n$

$x^* = \lim_{k\to\infty} x^{(k)}$, ϕ continuous $\Rightarrow x^* = \phi_f(x^*)$ ↳ fixed pt of ϕ_f

consistency: $\phi_f(x) = x \Leftrightarrow F(x) = 0$ ↳ many available! all yield same x^* for $k\to\infty$ (but cvg isn't the same)

How to predict cvg:



$\phi'(x^*) \in]-1,1[$ → convergence (local, linear) rate $\cong \phi'(x^*)$ (at least!)

$|\ "\ | \geq 1$ → divergence

$= 0$ → quadratic cvg!

slope of $\phi(x)$ at fixed pt crucial!

Analysis: Taylor expansion around x^*: $x^{(k+1)} - x^* = \phi(x^{(k)}) - \phi(x^*) = \phi'(x^*)(x^{(k)} - x^*) + \frac{1}{2}\phi''(x^*)(x^{(k)} - x^*)^2 + \dots$ (order if $|x^{(k)} - x^*| \ll 1$)

$D\phi(x^*)$ = jacobian $\in \mathbb{R}^{n,n}$

$\|\ \| < 1$ → local cvg, at least lin

global cvg: if $L: \sup_{x\in U} \|D\phi(x)\| < 1$: global linear (at least) cvg.

↳ the derivative is $\in]-1,1[$ everywhere, not only around x^*

$\phi: U \subset \mathbb{R}^n \to \mathbb{R}^n$

ZERO FINDING ($n=m=1$?)

seek $x^* \in \mathbb{R}$: $f(x^*) = 0$, $f: I \subset \mathbb{R} \to \mathbb{R}$ continuous

Bisection

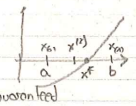
interval $[a,b]$, $f(a)<0$, $f(b)>0$ $\Rightarrow \exists x^* \in]a,b[: f(x^*) = 0$ (intermediate value theorem)

$|x^* - x^{(k)}| \leq 2^{-k}|b-a|$ ("kind of" linear cvg)

→ $\left\lceil \log_2 \left(\frac{|b-a|}{Tol}\right) \right\rceil$ necessary steps

⇒ robust method, $\varepsilon_k <$ Tol is guaranteed

⇒ simple, only eval of f required



Model function methods

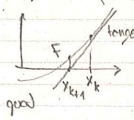
in step k: replace f with $\tilde{f}_k$ → $x^{(k+1)}$: $\tilde{f}_k(x^{(k+1)}) \overset{!}{=} 0$ (easier to find, analytical)

Newton's method

Assume f differentiable. $\tilde{F}_k(x) = f(x^{(k)}) + f'(x^{(k)})(x - x^{(k)})$ → tangent in $(x^{(k)}, F(x^{(k)}))$

⇒ fixed point iteration with $\phi(x) = x - \frac{F(x)}{F'(x)}$ $f'(x) \neq 0$

$\phi'(x) = 1 - \frac{F'(x)^2 - F(x)F''(x)}{F'(x)^2} = \frac{F(x)F''(x)}{F'(x)^2}$ $\phi'(x^*) = 0$ ⇒ local good cvg



ex $f(x) = x^2 - a$ → $x^* = \sqrt{a}$

NI = newton iteration

drawback: • local cvg (no global) • maybe $f'(x)$ not available

nodal analysis

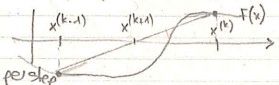
multi point method ($m>1$)

(es use last 2 points to find next)

Secant method

$\tilde{F}_k(x) = F(x^{(k)}) + \Delta s\,(x - x^{(k)})$, $\Delta s = \frac{F(x^{(k)}) - F(x^{(k-1)})}{x^{(k)} - x^{(k-1)}}$

$x^{(k+1)} = x^{(k)} - \frac{F(x^{(k)})}{\Delta s}$



advantages: • 1 f-evaluation per step • no derivatives → black box suitable

order of cvg can be fractional!

SM (secant method) $x^{(k+1)} = \phi(x^{(k)}, x^{(k-1)})$ $\phi(x,q) = x - \frac{x-q}{f(x)-f(q)}F(x)$

$F(x^*) = 0$, $\phi(x^*, x^*) = x^*$ $\frac{\partial\phi}{\partial x} = \frac{\partial\phi}{\partial q} = \frac{\partial^2\phi}{\partial x^2} = \frac{\partial^2\phi}{\partial q^2} = 0$

Asymptotic error recursion (taylor exp) $e^{k+1} = \phi(x^* + e^{(k)}, x^* + e^{(k-1)}) - x^* = k e^{(k)} e^{(k-1)} + \text{"small term"}$

$|e^{(k)}| \leq C\,|e^{(k-1)}|^p$ for order p

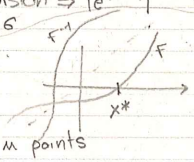
$|e^{(k+1)}| \cong C^{p+1}|e^{(k-1)}|^{p^2}$ ⇒ $p^2 - p - 1 = 0$ ⇒ $p \cong 1{,}6$

plug into recursion ⇒ $|e^{(k-1)}|^{p^2-p-1} \cong K^{-p}C = \text{const}$ $\forall k$

Inverse interpolation method

Assume $f: I \subset \mathbb{R} \to \mathbb{R}$ strictly monotone

Idea: $F(x^*) = 0 \Leftrightarrow f^{-1}(0) = x^*$



interpolate f^{-1} by a polynomial of deg $(m-1)$ in m points

$p(F(x^{(k-j)})) = x^{(k-j)}$ $j = 0 \dots m-1$ ↳ now $x^{(k+1)} = p(0)$

$m=2$ → secant method

$m=3$ → quadratic inv. interpolation

Asymptotic efficiency

↳ high accuracy ↳ $\gamma = \frac{\text{gain (extra correct digits)}}{\text{work}}$

work = #F-eval + #F'-eval $\cong w$ (per step)

gain: $\|e^{(k)}\| \leq \rho \|e^{(0)}\|$ → digits gained $-\log_{10}\rho$ ($0 < \rho < 1$)

$K(\rho)$ = minimal # of steps for given gain

efficiency $= \frac{1}{w \cdot K(\rho)} \frac{|\log\rho|}{} = \frac{\#\text{gained digits}}{\text{tot. work required}}$

→ Assume iteration of order $p>1$ $\|e^{(k)}\| \leq C\|e^{(k-1)}\|^p = C^{1+p+p^2+\dots+p^{k-1}}\|e^{(0)}\|^{p^k}$

som guaranteed, if $C^{\frac{p^k-1}{p-1}}\|e^{(0)}\|^{p^{k-1}} \lesssim \rho$ (p^k)

$L_0 := C^{\frac{1}{p-1}}\|e^{(0)}\| < 1$: $k \geq \frac{\log(\log\rho/\log L_0)}{\log p} + 1) \Rightarrow \log p \cdot k \gtrsim \log|\log\rho| - \log|\log L_0|$

$\lesssim \log|\log\rho|$

$\text{Eff}|_{\rho\to 0} = \begin{cases} \frac{-\log C}{w} & p = 1 \\ \frac{\log p}{w}\frac{|\log\rho|}{|\log\rho|} & p > 1 \end{cases}$

↳ independent, use for comparisons

es. Newton ($p=2$) vs Secant method ($p \cong 1{,}6$)

$\frac{\log 2}{2w_0}$: $\frac{\log(1{,}6)}{w_0} \cong 0{,}7$

↳ more efficient

Newton's method

find $x^* \in \mathbb{R}^n$: $F(x^*) = 0$, $F: D \subset \mathbb{R}^n \to \mathbb{R}^n$

As before, model function method based on local linearization

$F(x) \longrightarrow \tilde{F}_k(x) = F(x^{(k)}) + DF(x^{(k)})(x - x^{(k)})$

iteration: $x^{(k+1)} = x^{(k)} - DF(x^{(k)})^{-1}F(x^{(k)})$ ↳ solution $\in \mathbb{R}^{n,n}$

$x^{(k+1)} = \phi(x^{(k)})$, $\phi(x) = x - DF(x)^{-1}F(x)$

in C++ (see code) Objects of type VecType must provide a method

z = VecType operator (const VecType &x, const VecTypeRef f);

→ solves $DF(x)z = f$

Note: $n \gg 1$: computation of newton correction can be expensive, $C(n^3)$

→ Affine invariance of Newt. meth

$\Delta\ G_A(x) = A \cdot F(x)$, $A \in \mathbb{R}^{n,n}$ $x^{(k+1)} = x^{(k)} - (A\,DF(x^{(k)}))^{-1} A F(x^{(k)})$

$DG_A(x) = A \cdot DF(x)$ $DF(x^{(k)})^{-1}F(x^{(k)})$ ⇒ same N.I for all A

↳ termination criteria should say step at same time for all A

Case study: quasi-linear system of equations

$A(x)x = b$, LSE with solution dependent syst. matrix $A(x) = \begin{pmatrix} 3+\|x\| & 1 & \\ 1 & 3+\|x\| & 1 \\ & & \end{pmatrix}$

sought: Newt. meth for $f(x) = A(x)x - b \doteq 0$

$A(x)x = Tx + x\|x\|_2$, $T = \begin{pmatrix} 3 & 1 & \\ 1 & 3 & 1 \\ & & \end{pmatrix}$

derive: $D\{x \to Tx\}\xi = T \cdot 1 = T$

$D\{x \to x\|x\|\}\xi = D\{x \to [x_1\sqrt{x_1^2+\dots+x_n^2}, \dots, x_n\sqrt{\dots}]^T\}_{i=1}^n \cdot \xi$, jacobian by partial differentiation

product rule $D\{x \to F(x)\cdot G(x)\}h = DF(x)h \cdot G(x) + F(x)\,DG(x)(h)$ → si ottiene a quello che derivano

chain rule $D\{x \to F(G(x))\}h = DF(G(x))DG(x)h$

general derivatives → analysis $\phi(x+h) \overset{\text{extrapolation}}{=} \phi(x) + D\phi(x)h + \mathcal{O}(h)$ → tends to 0 faster than h

Apply this to $x \to x\|x\|$ 1) product rule $D\{x \to x\|x\|\}h = I\|x\| + x\,D\{x \to \|x\|\}h$

2) $\|x\| = \sqrt{\ } \circ x^Tx$ chain: $[F \to \sqrt{\ }, G \to x^Tx]$

$D\{x \to \|x\|h\} = \frac{1}{2\|x\|} \cdot (x^Th + h^Tx) = \frac{1}{\|x\|}x^Th = \|x\|h + x\frac{1}{\|x\|}x^Th = \underbrace{\left(\|x\|_2 I + \frac{xx^T}{\|x\|}\right)}_{\text{jacobian}}h$

$\Rightarrow DF(x) = \underbrace{T + \|x\|I}_{A(x)} + \underbrace{\frac{xx^T}{\|x\|}}_{\text{rank 1 matrix}}$ ↳ rank 1 $\Rightarrow$ rank 1 modification of $A(x)$

Convergence of Newton method

↳ locally & quadratically: $D\phi(x^*) = 0$, $D\phi(x)h = \dots = -D\{\dots\}h \cdot F(x)$

$D\phi(x) = I - D(D(x)^{-1})F(x) - DF(x)^{-1}DF(x)$

if ... then

if all these are satisfied:
- $D \subset \mathbb{R}^n$ open, convex (U)
- $F: D \to \mathbb{R}^n$ continuously different
- $DF(x)$ regular $\forall x \in D$

$\exists L \geq 0$:
$\exists x^*: f(x^*) = 0$
initial guess $x^{(0)}$ satisfies $\rho := \|x^* - x^{(0)}\| < \frac{2}{L} \wedge B_\rho(x^*) \subset D$

then these hold:
- $x^{(k)} \in B_\rho(x^*) := \{y \in \mathbb{R}^n \mid \|y - x^*\| < \rho\}$
- $\lim_{k\to\infty} x^{(k)} = x^*$
- $\|x^{(k+1)} - x^*\| \leq \frac{L}{2}\|x^{(k)} - x^*\|^2$ ↳ loc. quad. cvg

Termination of NI

$\varepsilon_{k+1} \ll \varepsilon_k \Rightarrow \|x^{(k+1)} - x^{(k)}\| \cong \varepsilon_k$ only few steps will suffice. each is very expensive

↳ posteriori termination

correction based termination: one step too much! ⇒ use simplified Newton correction

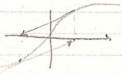
$\Delta\bar{x}^{(k)} = DF(x^{(k-1)})^{-1}F(x^{(k)})$ → $\mathcal{O}(n^2)$ effort

STOP if $\Delta\bar{x}^{(k)} \leq$ Tabs $\leq$ Trel $\cdot \|x^{(k)}\|$ Note: affine invariant, because take for $\Delta\bar{x}^{(k)}$

↳ before it was k → one computation too much

Damped Newton method

→ cvg really bad



overshooting newton corrections

Damping: $\lambda^{(k)} > 0$: $x^{(k+1)} := x^{(k)} - \lambda^{(k)} DF(x^{(k)})^{-1}F(x^{(k)})$

$0 < \lambda \leq 1$ damping factor

Affine invariant damping strategy (choice of factor)

→ choose maximal λ :

$\|\Delta\bar{x}(\lambda^{(k)})\| \leq \left(1 - \frac{\lambda^{(k)}}{2}\right)\|\Delta\bar{x}^{(k)}\|$ ↳ $= DF(x^{(k)})^{-1}F(x^{(k)})$ → current N. correction

↳ $= DF(x^{(k)})^{-1}F(x^{(k)} - x^{(k)}\Delta x^{(k)})$ → tentative simplified N. corr.

if fail ⇒ $\lambda \leftarrow \frac{\lambda}{2}$

else ⇒ 1-step with damp. factor λ, 2λ for next step

$\lambda \cong 0$ a bit of damping

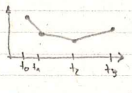
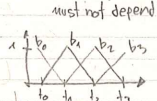
$\cong 1$ not much

Jacobian Matrix

$J = \frac{df}{dx} = \left[\frac{\partial f}{\partial x_1} \dots \frac{\partial f}{\partial x_n}\right] = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & & \\ & & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$

3. DATA INTERPOLATION IN 1D

Def: Given: data points $(t_i, y_i) \in \mathbb{R}^2$ $i=0,\dots,n$
Sought: interpolant $f: I \subset \mathbb{R} \to \mathbb{R}$: $f(t_i) = y_i$
Assumption: $t_0 < t_1 < \dots < t_n$
Interpolant can be represented as linear comb. of basis functions $b_j: I \to \mathbb{R}$ ↳ may depend on t_i must not depend on y_i
$f(t) = \sum_{j=0}^{n} c_j b_j(t)$ c_j parameters

piecewise linear interpolation
→ connect data point by segments
basis functions: (tent functions)



$f(t) = \sum_{j=0}^{n} y_j b_j(t)$, $f(t_i) = y_i$ because $b_j(t_i) = \begin{cases} 1 & i=j \\ 0 & \text{else} \end{cases} = \delta_{ij}$

$y(t) = \begin{cases} \frac{t - t_{i-1}}{t_i - t_{i-1}} & \dots \\ 0 \end{cases}$

linear interpolation
(is a linear mapping) $I: \begin{cases} \mathbb{R}^{n+1} \to C^0[t_0, t_n] \\ y = [y_i]_{i=0}^n \to f \end{cases}$

is called linear if $I(c_1 y + \beta z) = c I(y) + \beta I(z) \quad \forall y, z \in \mathbb{R}^{n+1}, c, \beta \in \mathbb{R}$

$A_c := \begin{bmatrix} b_0(t_0) & \dots & b_n(t_0) \\ \vdots & & \vdots \\ b_0(t_n) & & b_n(t_n) \end{bmatrix} \begin{bmatrix} c_0 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_0 \\ \vdots \\ y_n \end{bmatrix} =: y$ f exists, unique $\Leftrightarrow$ $n = n$

Global polynomial interpolation

$\mathcal{P}_n = \{ t \to p(t) := \sum_{j=0}^{n} a_j t^j, a_j \in \mathbb{R} \}$, degree $\le n$, $\dim \mathcal{P}_n = n+1$ (vector space)
↳ basis
advantages: • smooth C^∞ (easy to differentiate) • easy to evaluate • Taylor polyn.

Horner scheme (distributive evaluation)
$p(t) = t(t(\dots t(a_n t + a_{n-1}) + a_{n-2}) \dots) + a_0$ → effort $O(n)$

theory: Lagrange polyn. interpolation problem (same as above)
→ compute $p(t) \in \mathcal{P}_n$ with $p(t_i) = y_i$, $i = 0 \dots n$

$L_i(t) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{t - t_j}{t_i - t_j} \in \mathcal{P}_n$, $L_i(t_j) = \delta_{ij}$ $\Rightarrow$ $p(t) = \sum_{j=0}^{n} y_j L_j(t)$ → exists & unique

it's a mapping $I_T: \mathbb{R}^{n+1} \to \mathcal{P}_n$

Hermite interpolation problem
given t_i nodes $i=0\dots n$
y_i values
c_i slopes
find $p \in \mathcal{P}_{2n+1}$: $p(t_i) = y_i$, $p'(t_i) = c_i$ (existence & uniqueness)

Algorithms:
multiple evaluations we want to evaluate in many ($N \gg n$) points $x_1, \dots, x_N$
lagrange not efficient → each requires $O(n^2)$ → $O(N n^2)$
observation: $p(t) = \sum_{i=0}^{n} L_i(t) y_i = \sum_{i=0}^{n} \left(\prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{t - t_j}{t_i - t_j} \right) y_i = \sum_{i=0}^{n} \lambda_i \left(\prod_{\substack{j=0 \\ j \neq i}}^{n} (t - t_j) \right) y_i = \prod_{j=0}^{n} (t - t_j) \sum_{i=0}^{n} \frac{\lambda_i}{t - t_i} y_i$ ↳ due to $j \neq i$

with $p(t) = 1$, $y_i = 1$
$\Rightarrow \prod_{j=0}^{n} (t - t_j) = \frac{1}{\sum_{i=0}^{n} \frac{\lambda_i}{t - t_i}}$

$\lambda_i := \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{1}{t_i - t_j}$ ↳ no i dependence
↳ depends on t_i, compute in constructor

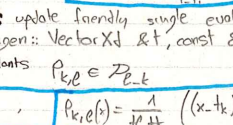
each evaluation is $O(n)$ → tot $O(n^2 + Nn)$

Barycentric formula
$p(t) = \frac{\sum_{i=0}^{n} \frac{\lambda_i}{t - t_i} y_i}{\sum_{i=0}^{n} \frac{\lambda_i}{t - t_i}}$

Aitken-Neville: update friendly single evaluation → (most efficient for single eval)
double eval (const Eigen::VectorXd &t, const Eigen::VectorXd &y, double x)
partial lagrange interpolants $p_{k,\ell} \in \mathcal{P}_{\ell - k}$ ↳ evaluate here

recursion: $p_{k,k} = y_k$, $p_{k,\ell}(x) = \frac{1}{t_\ell - t_k} \left((x - t_k) p_{k+1,\ell}(x) - (x - t_\ell) p_{k,\ell-1}(x) \right)$ unique & satisfies conditions

effort: $O(n^2)$



→ see code

suitable for simultaneous eval + adding data pts

Extrapolation to zero

objective: compute $\lim_{h \to 0} \Psi(h)$ (smooth, but hard to evaluate for $|h| \ll 1$, $h < \sqrt{\text{eps}}$ (cancelation)

eg. difference quotient $\Psi(h) = \frac{f(x+h) - f(x)}{h}$

idea: • pick large $h_0, \dots, h_m$
• evaluate $\Psi(h_i)$
$\Rightarrow \Psi(0) \approx p(0)$ with interpolating polynomial $p \in \mathcal{P}_n$, $p(h_i) = \Psi(h_i)$

AN: x has to be known in advance

newton basis $N_0(t) := 1$, $N_k(t) = \prod_{\ell=0}^{k-1} (t - t_\ell) \in \mathcal{P}_k$ (leading coeff = 1)

$p = \sum_{j=0}^{n} a_j N_j$ (lagrange interpolant) $N_k[t_j] := 0$, $j = 0 \dots k-1$ LSE: $\begin{bmatrix} 1 & & & \\ 1 & (t_1 - t_0) & & \\ \vdots & \vdots & \ddots & \\ 1 & (t_n - t_0) & \dots & \prod_{i=0}^{n-1} (t_n - t_i) \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ \vdots \\ y_n \end{bmatrix}$

$p_{k,\ell} = \dots = p_{k+1,\ell}(x) + \frac{x - t_\ell}{t_\ell - t_k} p_{k,\ell-1}(x)$ $\begin{vmatrix} a_{k,\ell} = \frac{1}{t_\ell - t_k} (a_{k+1,\ell} - a_{k,\ell}) \\ a_{k,k} = y_k \end{vmatrix}$

divided difference recursion → see code
→ represent version of horner scheme $p(x) = ((a_n (x - t_n) + a_{n-1})(x - t_{n-1}) + a_{n-2}) + \dots + a_0$

Sensitivity: given t_i, y_i (might be perturbed) → output $p(t)$
needed: norms on data space $\mathbb{R}^{n+1}$ → $\|y\|_\infty = \max_i |y_i|$

lagrange interp. is linear $L: X \to Y$ linear: $L(x + \delta x) = L(x) + L(\delta x)$

Amplification of perturbation: $\sup_{\delta x \in X \setminus \{0\}} \frac{\|L(\delta x)\|_Y}{\|\delta x\|_X} =: \|L\|_{X \to Y}$

$\|I_T(y)\|_\infty = \left\| \sum_{j=0}^{n} y_j L_j \right\|_\infty \le \max_j |y_j| \left\| \sum_{j=0}^{n} |L_j| \right\|_\infty \le \|y\|_\infty \left\| \sum_{j=0}^{n} |L_j| \right\|_\infty$, $y \neq 0$

$\Rightarrow \|I_T\|_{\infty \to \infty} \le \left\| \sum_{j=0}^{n} |L_j| \right\|_\infty$ → **Lebesgue constant** (measure of how good the interpolation is)

$\lambda_T \ge C e^{n/2}$ lower bound

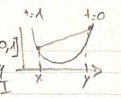
SHAPE PRESERVING INTERPOLATION

given: (t_i, y_i) $i=0,\dots,n$ $t_0 < t_1 < \dots < t_n$ find f as usual

properties
• sign
• monotonicity (t_i, y_i) monotone $\Leftrightarrow$ $y_i \ge y_{i-1}$ or $y_i \le y_{i-1}$ $\forall i = 1 \dots n$
• curvature ∪ convex $\Leftrightarrow$ $\Delta_j \le \Delta_{j+1}$ $j = 1 \dots n-1$
∩ concave $\Leftrightarrow$ $\ge$

$\Delta_j := \frac{y_j - y_{j-1}}{t_j - t_{j-1}}$ (local slope)

$f: I \subset \mathbb{R} \to \mathbb{R}$ convex $\Leftrightarrow$ $f(tx + (1-t)y) \le t f(x) + (1-t) f(y)$ $t \in [0,1]$, $\forall x, y \in I$



shape preservation
Local shape properties of data ⇒ shape properties of interpolant
↳ holds on all $[t_{k-m}, t_k]$

⇉ lagrange interp → NO shape preservation
⇉ piecewise linear → YES (perfect)

▷ S.t. $C[t_0, t_n]$ be piecewise linear interpolant of $(t_i, y_i) \in \mathbb{R}^2$ for every subinterval $I = [t_j, t_k]$
if $(t_i, y_i)|_I$ are • positive/negative ⇒ s_I is • positive/negative
• monotonic (inc/dec) ⇒ • monotonic (inc/dec)
• convex/concave ⇒ • convex/concave

Locality: change of y_i only affects $[t_{i-1}, t_{i+1}]$ (this is not the case with lagrange)
drawback: only C^0!

PIECEWISE CUBIC HERMITE INTERPOLATION

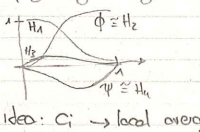
given (t_j, y_j), $j = 0 \dots n$, slopes c_j
⇒ piecewise cubic herm. interpolant $s: [t_0, t_n] \to \mathbb{R}$ is defined by requirements
$s|_{[t_{i-1}, t_i]} \in \mathcal{P}_3$, $i = 1 \dots n$ $s(t_i) = y_i$, $s'(t_i) = c_i$ ($s \in C^1$)

$t \in [t_{i-1}, t_i]$

⇒ express s locally using basis $s(t) = y_{i-1} H_1(t) + y_i H_2(t) + c_{i-1} H_3(t) + c_i H_4(t)$

	$H(t_{i-1})$	$H(t_i)$	$H'(t_{i-1})$	$H'(t_i)$
H_1	1	0	0	0
H_2	0	1	0	0
H_3	0	0	1	0
H_4	0	0	0	1

$H_1(t) = \phi\left(\frac{t_i - t}{h_i}\right)$ $H_3(t) = -h_i \psi\left(\frac{t_i - t}{h_i}\right)$ $h_i := t_i - t_{i-1}$
$H_2(t) = \phi\left(\frac{t - t_{i-1}}{h_i}\right)$ $H_4(t) = h_i \psi\left(\frac{t - t_{i-1}}{h_i}\right)$ $\phi(\tau) = 3\tau^2 - 2\tau^3$
$\psi(\tau) = \tau^3 - \tau^2$

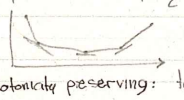


task: find slopes from values: $c_i = c_i(y_0, \dots, y_n)$
→ Natural: $c_i = c_i(y_{i-1}, y_i, y_{i+1})$

idea: c_i → local average of slopes

equispaced nodes: $c_i = \frac{1}{2}(\Delta_i + \Delta_{i+1})$

$c_i = \begin{cases} \Delta_1 & \text{if } i = 0 \\ \Delta_n & \text{if } i = n \\ \frac{t_{i+1} - t_i}{t_{i+1} - t_{i-1}} \Delta_i + \frac{t_i - t_{i-1}}{t_{i+1} - t_{i-1}} \Delta_{i+1} & \text{else} \end{cases}$



⇒ doesn't preserve monotonicity!

$\Delta_i = \frac{y_i - y_{i-1}}{t_i - t_{i-1}}$

monotonicity preserving: to avoid overshoot, use **slope limiting**

$c_i = \begin{cases} 0 & \text{if } \operatorname{sgn}(\Delta_i) \neq \operatorname{sgn}(\Delta_{i+1}) \\ \text{some average of } \Delta_i, \Delta_{i+1} & \text{else} \end{cases}$

Harmonic mean (weighted) $c_i = \frac{w_i + w_{i+1}}{\frac{w_i}{\Delta_i} + \frac{w_{i+1}}{\Delta_{i+1}}}$, $(w_i + w_{i+1} = 1)$

good choice of weights: $w_a := w_i = \frac{2h_{i+1} + h_i}{3(h_{i+1} + h_i)}$ $w_b := w_{i+1} = \frac{h_{i+1} + 2h_i}{3(h_i + h_{i+1})}$ $\Rightarrow c_i = \frac{3(h_{i+1} + h_i)}{\frac{2h_{i+1} + h_i}{\Delta_i} + \frac{h_{i+1} + 2h_i}{\Delta_{i+1}}}$

$\Rightarrow$ this is still non linear, but preserves monotonicity & $\in C^1$

MATLAB: pchip (t, y, x) – piecewise cubic herm. ip.

SPLINES

(piecewise polynomial C^k-interpolants, $k \ge 2$ perceived as smooth)

Spline space: given $I = [a, b] \subset \mathbb{R}$ and mesh/nodes set $M := \{a = t_0 < t_1 < \dots < t_n = b\}$ the vector space $S_{d,M}$ of the spline functions of deg d, order $d+1$ is

$S_{d,M} := \{ s \in C^{d-1}(I) : s_j = s|_{[t_{j-1}, t_j]} \in \mathcal{P}_d \ \forall j = 1 \dots n \}$
↳ $d-1$ times continuously differentiable ↳ local polyn deg d

$d = 0$ → p.w. const, discontinuous → step function
$d = 1$ → p.w. linear, polygon

dimension $(S_{d,M}) = \underbrace{n \cdot \dim \mathcal{P}_d}_{\text{dimension of piece-wise polyn}} - \underbrace{(n-1) \cdot d}_{\#\text{ continuity constraints } (C^{d-1})} = n + d$

$s \in S_{d,M} \Rightarrow s' \in S_{d-1,M}$
$\int s dt \in S_{d+1,M}$

• cubic spline interp ($d = 3$)
given: $M := \{t_i\}_{i=0}^n$, values $y_0 \dots y_n$, find $s \in S_{3,M}$: $s(t_i) = y_i$ ($n+1$ conditions ↔ $\dim S_{3,M} = n+3$)
▷ 2 DOF (degree of freedom) remain

$s|_{[t_{j-1}, t_j]}(t) = s(t_{j-1}) \cdot (1 - 3\tau^2 + 2\tau^3) + s(t_j) \cdot (3\tau^2 - 2\tau^3) + h_j s'(t_{j-1}) \cdot (\tau - 2\tau^2 + \tau^3) + h_j s'(t_j) \cdot (-\tau^2 + \tau^3)$
unknown slopes c_j
$h_j = t_j - t_{j-1}$
$\tau = \frac{t - t_{j-1}}{h_j}$
equations from C^2-continuity

$s(t_j) = y_j$
$s'(t_j) = c_j$

$s''|_{[t_{j-1}, t_j]}(t_{j-1}) = -6 \cdot s(t_{j-1}) h_j^{-2} + 6 s(t_j) \cdot h_j^{-2} - 4 h_j^{-1} s'(t_{j-1}) - 2 \cdot h_j^{-1} s'(t_j)$
t_{j-1}: $+6$, -6, $+2$, $+4$
t_j:

$\frac{1}{h_j} c_{j-1} + \left(\frac{2}{h_j} + \frac{2}{h_{j+1}} \right) c_j + \frac{1}{h_{j+1}} c_{j+1} = 3 \left(\frac{y_j - y_{j-1}}{h_j^2} + \frac{y_{j+1} - y_j}{h_{j+1}^2} \right)$

→ matrix form

$\begin{bmatrix} 2/h_1 & 1/h_1 & & & \\ 1/h_1 & 2(1/h_1 + 1/h_2) & 1/h_2 & & \\ & \ddots & \ddots & \ddots & \\ & & 1/h_{n-1} & 2(1/h_{n-1} + 1/h_n) & 1/h_n \\ & & & 1/h_n & 2/h_n \end{bmatrix} \begin{bmatrix} c_0 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} 3\frac{y_1 - y_0}{h_1^2} \\ 3\left(\frac{y_1 - y_0}{h_1^2} + \frac{y_2 - y_1}{h_2^2} \right) \\ \vdots \\ 3\left(\frac{y_{n-1} - y_{n-2}}{h_{n-1}^2} + \frac{y_n - y_{n-1}}{h_n^2} \right) \\ 3\frac{y_n - y_{n-1}}{h_n^2} \end{bmatrix}$

$b_i := \frac{1}{h_{i+1}}$ $a_i := 2(b_i + b_{i-1})$ $b_i, a_i > 0$
⇒ strictly diagonally dominant ⇒ invertible

fix missing conditions
• COMPLETE CSI prescribe $s'(t_0) = c_0$, $s'(t_n) = c_n$ (skip first & last col of A)
• NATURAL CSI requires $s''(t_0) = s''(t_n) = 0$ (2 extra equations)
• PERIODIC CSI $s'(t_0) = s'(t_n)$ $(c_0 = c_n)$
$s''(t_0) = s''(t_n)$ (1 extra equation)

2 extra eq: $\frac{2}{h_1} c_0 + \frac{1}{h_1} c_1 = 3 \frac{y_1 - y_0}{h_1^2}$
$\frac{1}{h_n} c_{n-1} + \frac{2}{h_n} c_n = 3 \frac{y_n - y_{n-1}}{h_n^2}$

→ finally $n \times n$ LSE

properties
• minimal bending [E]: $\mathcal{E}_{bb}(f) = \int_a^b |f''(\tau)|^2 d\tau$
spline - minimizes: $\mathcal{E}_{bb}(s) \le \mathcal{E}_{bb}(f)$ $\forall f \in C^2[a,b]$, $f(t_i) = y_i$ → see proof
• NO monotonicity preservation
• NO locality (CSI linear ⇒ gauge locality by looking at cardinal interpolant $I y e_j$)
CSI is "closest local": cardinal interp decays exponentially

4. APPROXIMATION OF FUNCTIONS IN 1D

focus: $n=d=1$

given: $f: D\subset\mathbb{R}^n \to \mathbb{R}^d$ (often procedural form $y=\text{feval}(x)$)
goal: find "simple" function $\tilde f: D\to\mathbb{R}^d$ such that approx. error $f-\tilde f$ is small
↳ piecewise polyn ↳ example, small $\|f-\tilde f\|$

norms: supremum $\|f\|_{L^\infty(D)} := \sup_{x\in D}|f(x)|$
L^2-norm $\|f\|^2_{L^2(D)} := \int_D |f(x)|^2 dx$

approx. scheme $A: X\to V$ (X,V spaces of functions) eg $X=C^0(I)$, $I\subset\mathbb{R}$, $V=\mathcal{P}_n$ or $S_{3,M}$

model reduction: see example

Approx by interpolation: interp + sampling → approx. scheme
$f: I\subset\mathbb{R}\to\mathbb{K}$ —sampling→ $(t_i, y_i=f(t_i))$ —interp scheme→ $\tilde f = I_T y$ $(\tilde f(t_i)=y_i)$
(advantage: free choice of nodes)

Approx by global polyn:

Taylor: $f(t) = \sum_{j=0}^{n} \frac{f^{(j)}(t_0)}{j!}(t-t_0)^j + f^{(n+1)}(\xi)\frac{(t-t_0)^{n+1}}{(n+1)!}$
$\in\mathcal{P}_n$; remainder, small for $|t-t_0|\ll 1$

polyn can uniformly approximate any $f\in C^k(I)$, along with their first k derivatives

Bernstein approximant

$f\in C^0[0,1] \to$ n^{th} approx $\boxed{B_n(f)(t) = \sum_{j=0}^{n} f\left(\frac{j}{n}\right)\binom{n}{j} t^j (1-t)^{n-j} \in \mathcal{P}_n}$
$\|f-B_n\|_\infty \to 0$ for $n\to\infty$. If $f\in C^m[0,1] \Rightarrow \|f^{(k)} - B_n^{(k)}\|_\infty \to 0$ for $n\to\infty$, $k=0\dots m$

Best approximation

: size of (best approx) of $f\in X$ in $\mathcal{P}_k$ with norm $\|\cdot\|$ is
$\boxed{\text{dist}_{\|\cdot\|}(f,\mathcal{P}_k) := \inf_{p\in\mathcal{P}_k}\|f-p\|}$

bound: $f\in C^r[-1,1]$ → $\inf \le (1+\pi^2/2)^r \frac{(k-r)!}{k!}\|f^{(r)}\|_{L^\infty[-1,1]}$; $n\to\infty \Rightarrow \mathcal{O}(n^{-r})$
↳ crucial role for smoothness

generalization to $[a,b]$: affine hf $\phi: \begin{cases}[-1,1]\to[a,b] \\ t\to a+\frac{1}{2}(b-a)(t+1)\end{cases}$

pullback $\phi^*: \begin{cases} C^0[a,b]\to C^0[-1,1] \\ f \to \{t\to f(\phi(t))\}\end{cases}$

Note: • $\phi^*: \mathcal{P}_n\to\mathcal{P}_n$ bijective & linear
• $\|\phi^* f\|_{L^\infty[-1,1]} = \|f\|_{L^\infty[a,b]}$
• $\|(\phi^* f)^{(k)}\|_{L^\infty[-1,1]} = \left(\frac{1}{2}(b-a)\right)^r \|f^{(k)}\|_{L^\infty[a,b]}$

chain rule $(\phi^* f)' = (f\circ\phi)' = (f'\circ\phi)\phi'$

error estimates for poly interp

interp scheme ⇒ approx scheme
↳ given on $[-1,1]$ → generalize on $[a,b]$ using ϕ

lagrangian approx $L_T: \begin{cases} C^0(I)\to\mathcal{P}_n \\ f\to I_T(y),\ y=(f(t_0),\dots,f(t_n))^T\in\mathbb{K}^{n+1}\end{cases}$

equispace nodes $T_n := \{t_j^{(n)} := a+(b-a)\frac{j}{n}\}$

convergence $\|f-L_T f\|\le T(n)$ asymptotically for $n\to\infty$
↳ error
if $\exists C\neq C(n): \|f-L_T f\|\le C\cdot T(n)$ for $n\to\infty$, $T(n)$ = bound of norm of int. error

algebraic conv:	$\exists p>0:\ T(n)\le n^{-p}$	(rate $p>0$)
exponential conv:	$\exists 0<q<1:\ T(n)\le q^n$	

→ 2 classes for the error bounds are sharp (no larger p / smaller q available)

eg: $\varepsilon_j \le Cn_j^{-p} \Rightarrow \log\varepsilon_j \le \log C - p\log n_j$
$\exp: \varepsilon_j \le Ce^{-Bn_j} \Rightarrow \log\varepsilon_j \le \log C - Bn_j$

lagrangian interp won't always converge → eg. oscillation (error → ∞)

▷ $f\in C^{n+1}(I)$. $\forall t\in I\ \exists \tau_t\in]\min\{t,t_0,\dots,t_n\}, \max\{t,t_0,\dots,t_n\}[$ such that

$$f(t)-L_T(f)(t) = \frac{f^{(n+1)}(\tau_t)}{(n+1)!}\prod_{j=0}^{n}(t-t_j)$$

→ proof

Rolle's theorem φ' has $n+1$ distinct zeros
$\varphi^{(n+1)}$ has at least 1 zero

$$\|f-L_T f\|_{L^\infty(I)} \le \frac{\|f^{(n+1)}\|_{L^\infty(I)}}{(n+1)!}\cdot\max_{t\in I}|(t-t_0)\cdots(t-t_n)| = \frac{\|f^{(n+1)}\|_{L^\infty}}{(n+1)!}\cdot(h^{(n+1)})$$

↳ smoothness of f essential
bound on error can blow up as $n\to\infty$
$n=1$ → linear, $=2$ → quadratic

Analicity of complex valued fk

$D\subset\mathbb{C}$ open set in complex plane. $f: D\to\mathbb{C}$ is analytic/holomorphic in D, if $f\in C^\infty(D)$ and it has convergent taylor series $\forall z\in D$
→ extension to $\mathbb{C}$: reinterpret $f: I\subset\mathbb{R}\to\mathbb{R}$ as $f: D\subset\mathbb{C}\to\mathbb{C}$

eg $e^z, \sin z, \cos z$: analytic in $\mathbb{C}$
$f(z)=\frac{1}{1+e^z}$: analytic in $\mathbb{C}\setminus\{(2k+1)\pi i, k\in\mathbb{Z}\}$
eg $f(t)=\frac{1}{1+t^2}$ } save formula
$f(z)=\frac{1}{1+z^2}$ } Analytic in $\mathbb{C}\setminus\{\pm i\}$

Analicity of interpoland

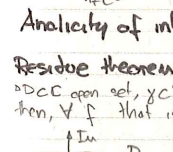
assume we can extend $f: I\to\mathbb{C}$ to $f: D\subset\mathbb{C}\to\mathbb{C}$ which is analytic on an open set D

Residue theorem

▷ $D\subset\mathbb{C}$ open set, $\gamma\subset D$ simple closed smooth curve, contractible in D, $\pi\subset D$ a finite set enclosed by γ. then, $\forall f$ that is analytic in $D\setminus\pi$ holds

$$\frac{1}{2\pi i}\int_\gamma f(z)dz = \sum_{p\in\pi}(\text{res}_p f)$$

↳ residual of f in $p\in\mathbb{C}$



contour integral $\int_\alpha^\beta f(z)dz = \int_\alpha^\beta f(\gamma(\tau))\frac{d\gamma}{d\tau}d\tau$ $(\gamma:[\alpha,\beta]\to\mathbb{C})$

$$\left|\int_\gamma f(z)dz\right| \le |\gamma|\cdot\max_{z\in\gamma}|f(z)|$$
↳ length

residue formula ▷ g,h complex functions, analytic in neighborhood $p\in\mathbb{C}$, $h(p)=0$, $h'(p)\neq 0$. then $\text{res}_p\frac{g}{h} = \frac{g(p)}{h'(p)}$

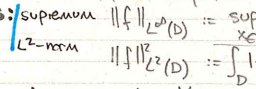
$$L_j(t) = \dots = \frac{\omega(t)}{(t-t_j)\omega'(t_j)} \qquad \omega(z)=(z-t_0)\cdots(z-t_n)\in\mathcal{P}_{n+1}$$

$$f(t) = -\sum_{j=1}^{n} f(t_j)\frac{\omega(t)}{(t_j-t)\omega'(t_j)} + \frac{\omega(t)}{2\pi i}\int_\gamma g_t(z)dz$$
(lagr. polyn; interp error; polyn. interpolant)

$$|f(t)-L_T(f)| \le \left|\frac{\omega(t)}{2\pi i}\int_\gamma \frac{f(z)}{(z-t)\omega(z)}dz\right| \le \frac{|\gamma|}{2\pi}\frac{\max_{a\le\tau\le b}|\omega(\tau)|}{\min_{z\in\gamma}|\omega(z)|}\cdot\frac{\max_{z\in\gamma}|f(z)|}{\text{dist}([a,b],\gamma)} \to = \frac{\inf_{z\in\gamma}}{a\le\tau\le b}|z-t|$$

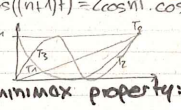
CHEBYCHEV INTERPOLATION

goal: optimal placement of nodes t_j for lagrange interp
$\|f-L_T(f)\|_{L^\infty(I)} \le \frac{1}{(n+1)!}\|f^{(n+1)}\|_{L^\infty(I)}\cdot\|\omega\|_{L^\infty(I)}$, $\omega(t)=\prod_{j=0}^{n}(t-t_j)$
→ choose $t_0,\dots,t_n$ such that $\|\omega\|_{L^\infty}$ is minimal ⇒ same as finding $q\in\mathcal{P}_{n+1}$ with
• leading coeff = 1
• minimizing $\|q\|_{L^\infty(I)}$
⇒ then t_j are zeros of q



n^{th} chebychev polyn is $T_n(t) := \cos(n\arccos(t))$ $|T_n(t)|\le 1$
3 term recursion: $T_{n+1}(t) = 2tT_n(t) - T_{n-1}(t)$ $T_0=1,\ T_1=t,\ T_n\in\mathcal{P}_n$, leading coeff $=1$

$\cos((n+1)t) = 2\cos t\cdot\cos nt - \cos(n-1)t$



zeros: $n\arccos t\in(2j+1)\frac{\pi}{2} \Rightarrow t_j = \cos\left(\frac{2j+1}{n}\frac{\pi}{2}\right)$ $j=0\dots n-1$ $\in\ ]-1,1[$
optimal nodes

minimax property: T_n polynomials minimize supremum norm L^∞:
$\|T_n\|_{L^\infty[-1,1]} = \inf\{\|p\|_{L^\infty[-1,1]} : p\in\mathcal{P}_n, p(t)=2^{n-1}t^n+\dots\}$ → proof
→ optimal choice $\omega = 2^{-n}T_{n+1}$

, nodes → they cluster at endpoints of $[-1,1]$ → see distribution

nodes: chebychev nodes on $[a,b]$: $t_k = a+\frac{1}{2}(b-a)\left(\cos\left(\frac{2k+1}{2(n+1)}\pi\right)+1\right)$ $k=0\dots n$ it's the affine transformation
$\|\omega\|_{L^\infty[a,b]} = 2^{-n}$
$\|\omega\|_{L^\infty[-1,1]} \le 2^{-n}$ $\|f-L_T(f)\|_{L^\infty} \le \frac{2^{-n}}{(n+1)!}\|f^{(n+1)}\|_{L^\infty}\cdot\left(\frac{b-a}{2}\right)^{n+1}$

sensitivity
lebesgue const λ_T → exponential blowup for equidistant nodes — connects interpolation to best approx
$\lambda_{\text{cheb}} \le \frac{2}{\pi}\log(1+n)+1$ (theoretical estimate)

$$\|f-L_T f\| \le (1+\lambda_T)\inf_{p\in\mathcal{P}_n}\|f-p\|_\infty$$
↳ interpolation ↳ norm of best approx error → see $\mathcal{O}(n^{-r})$ algebraic conv

$\gamma := \{z=\cos(\theta-z\log\rho),\ \theta\in[0,2\pi]\} = \dots = \{z=\frac{1}{2}(\rho+\rho^{-1})\cos\theta + i\frac{1}{2}(\rho-\rho^{-1})\sin\theta\}$
Re, Im
$\gamma(\theta) = A\cos\theta + B\sin\theta$ → ellipsis
→ larger ρ, larger ellipses, faster conv

lower bound of $|\omega|$ ↔ $|T_{n+1}|$ on γ

$$\|f-L_T f\|_{L^\infty[-1,1]} \le \frac{4|\gamma|}{\pi}\underbrace{\frac{1}{(\rho^{n+1}-1)(\rho+\rho^{-1}-2)}}_{\mathcal{O}(\rho^{-n}),\ \rho>1}\cdot\max_{z\in\gamma}|f(z)|$$

Admissible ρ: $\gamma_\rho\subset D$ (domain of analicity)
Interior of γ_ρ must be contained in D

MEAN SQUARE BEST APPROXIMATION

goal: compute best approximants
$X := \mathbb{R}$-vector space ↔ function space
$V\subset X$ finite dim subspace with basis $\{b_1,\dots,b_N\}$
sought: $q := \inf_{p\in V}\|f-p\|_x^2$ for $f\in X$ ↳ space of polyn
$(\cdot,\cdot)_X :=$ inner prod on X ⇒ $\|v\|_X = (v,v)_X^{1/2}$

$$\|f-p\|_x^2 = \|f\|^2 - 2(f,p)_x + \|p\|_x^2 = \|f\|^2 - 2\sum_j \gamma_j(f,b_j)_x + \sum_j\sum_k \gamma_j\gamma_k(b_j,b_k)_x$$
$$= \|f\|^2 - 2\,d^T c + c^T M c =: \phi(c)$$

$p = \sum_{j=1}^N \gamma_j b_j$, $d := [(f,b_j)_x]_{j=1}^N \in\mathbb{K}^N$, $M := [(b_j,b_k)_x]_{j,k=1}^N \in\mathbb{K}^{N,N}$, $c := [\gamma_j]_{j=1}^N \in\mathbb{K}^N$

$\phi: \mathbb{R}^N\to\mathbb{R}$, $\phi\in C^\infty$
⇒ find $c\in\mathbb{R}^N$: $\phi(c) = \min$
grad $\phi(c) = -2d+2Mc \doteq 0$
Hess $\phi(c) = M$
$x := \sum\xi_i b_i \in X$: $x^T M x = \|\sum\xi_i b_i\|_x^2 > 0 \Leftrightarrow x\neq 0$ → M s.p.d!
solve $\boxed{c = M^{-1}d}$

geometric view
best approximant by orthogonal projection
→ if q is the best approximant of f in V, then $f-q$ is orthogonal to every $p\in V$
$(f-q,p)_x = 0\ \forall p\in V \leftrightarrow f-q\perp q$
↳ best approx error

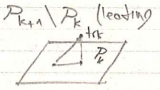
orthonormal basis $\{b_1,\dots,b_N\}$ ONB of N-dim vector space $V \Leftrightarrow (b_k,b_j) = \delta_{kj}$
↳ use gram-schmidt to compute them

polynomial mean square best approx $q = \sum_{j=1}^N (f,b_j)_x b_j$

$X = C^0[a,b]$, $V = \mathcal{P}_m$ $(f,g)_x = \int_a^b f(t)g(t)dt$

orthogonal poly:
sequence $r_0, r_1, \dots, r_m$ provides orthogonal polynomials with respect to $(\cdot,\cdot)_x$ if
$r_\ell\in\mathcal{P}_\ell$, $(r_k,r_\ell)_x = \delta_{k\ell}$
→ just orthogonal if $(r_k,r_\ell)_x = 0$ for $k\neq\ell$
eg: $\{r_0,\dots,r_m\}$ ONP (lin. indep) $t\cdot r_k\in\mathcal{P}_{k+1}\setminus\mathcal{P}_k$ (leading coeff $\neq 0$)

$r_{k+1}\in$ Span $\{tr_k, r_k, r_{k-1},\dots,r_0\}$
$\perp$ Span $\{r_k, r_{k-1},\dots,r_0\}$



$\tilde r_{k+1} = \frac{\tilde r_{k+1}}{\|\tilde r_{k+1}\|}$; $\tilde r_{k+1} = tr_k - \sum_{j=1}^k (tr_k, r_j)_x r_j$ (orthogonal projection)

Assume: $(tf,g)_x = (f,tg)_x\ \forall f,g$
$(tr_k, r_j)_x = (r_k, tr_j)_x = 0\ \forall j\le k-2$ (orthogonality)

→ sum becomes **3-term recursion**

$r_{-1}=0$, $r_0=1$, $r_{k+1}(t) := (t-\alpha_{k+1})r_k(t) - \beta_k r_{k-1}(t)$, $k=0\dots m-1$
then $r_k\in\mathcal{P}_k$, leading coeff $=1$, $\{r_0,r_1,\dots,r_m\}$ is ONB of $\mathcal{P}_m$

$\alpha_{k+1} = \frac{(tr_k - r_k(t)\|r_k\|^2, r_k)_x}{\|r_k\|_x^2}$
$\beta_k = \frac{\|r_k\|_x^2}{\|r_{k-1}\|_x^2}$

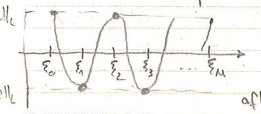
UNIFORM BEST-APPROXIMATION

given: $f \in C^0[a,b]$, polyn-degree n

sought: $q := \operatorname{argmin}_{p \in P_n} \|f-p\|_{L^\infty(I)}$ — it exists $\Leftrightarrow n+2$ points $a \le \xi_0 < \xi_1 < \dots < \xi_{n+1} \le b$

such that $|e(\xi_j)| = \|e\|_{L^\infty}$ $(j=0\dots n+1)$, $e(\xi_j) = -e(\xi_{j+1})$

with error $e := f-q$ → **chebychev alternation theorem**



Remez alg: CAT $\Rightarrow$ (+) $q(\xi_k) + (-1)^k \delta = f(\xi_k)$, $k=0\dots n+1$ — after choosing a basis for P_n → $(n+2)\times(n+2)$ LSE; unknowns = expansion coeffs of q & δ

- start from approximate alternants $\xi_j^{(0)}$, $j=0\dots n+1$
- solve (+) → q , • check sampling $\|f-q\|_\infty$ → STOP if small enough
- new set of alternants := extrema of $f-q$

APPROXIMATION BY PIECEWISE POLYN (with regard to mesh M on $[a,b]$)

$M = \{a = x_0, x_1, \dots, x_M = b\}$, interval/cell $[x_{j-1}, x_j]$

recall: cubic splines, cubic hermite, pw linear. advantage: cost $O(N = \#\text{of data points})$

- **piecewise lagrange interp**
  - choose local degree $n_j \in \mathbb{N}_0$ for each cell
  - choose set of local interp. points $T^j := \{t_0^j, \dots, t_{n_j}^j\} \subset I_j := [x_{j-1}, x_j]$ for each cell
  - define piecewise polyn. interpolant $s: [x_0, x_M] \to \mathbb{K}$

  $s_j := s_{|I_j} \in P_{n_j}$, $s_j(t_i^j) = f(t_i^j)$ — s_j is well defined (unique)

  disadvantage: $s \in C^0$ (in general $\notin C^1$)

focus: Asymptotic conv in maximum norm: **h-convergence**: $n_j = n$ $\forall j$ (fixed local deg)

consider sequence of meshes M_k with meshwidth $h_k := \max_j |x_j^{(k)} - x_{j-1}^{(k)}| \to 0$ $(k \to \infty)$

find bound $\|f - I_M f\|_{L^\infty[a,b]} \le C T(h_k)$ — study asymptotics for $h_k \to 0$

(I_M: interp. operator)

p-convergence: M fixed, raise local degree

- **pw cubic hermite interp**

given $f \in C^1[a,b]$, M, the pw. cubic Hermite interpolant (with exact slopes) $s: [a,b] \to \mathbb{R}$ is

$s|_{[x_{j-1}, x_j]} \in P_3$, $j=1\dots m$, $s(x_j) = f(x_j)$, $s'(x_j) = f'(x_j)$. — $s \in C^1$

convergence (h-conv) ▷ s be chip of $f \in C^4[a,b]$ on mesh M. then

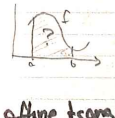
$$\|f-s\|_{L^\infty[a,b]} \le \frac{1}{4!} h_M^4 \|f^{(4)}\|_{L^\infty} \quad \text{with } h_M := \max_j |x_j - x_{j-1}|$$

- **cubic spline interpolation** (→ see ex)

rule of thumb: pw approx with polyn of deg p, h-convergence:
→ algebraic, max rate $p+1$ (if f smoothness required!)

5. NUMERICAL QUADRATURE

task: compute $\int_a^b f(t)\,dt$



cost := #f-eval

f given in procedural form (se analytical, only f.eval(x))

quadrature formula:

n-point quad-formula is approximation of integral using weighted sum

$$\int_a^b f(t)\,dt \approx Q_n(f) := \sum_{j=1}^n \omega_j^n f(c_j^n)$$

ω_j^n: quad. weights, c_j^n: — nodes

affine transformation

$\Phi: [-1,1] \to [a,b]$, $t \mapsto a + \frac{1}{2}(b-a)(t+1)$

$$\int_a^b f(t)\,dt = \int_{-1}^1 f(\Phi(t)) \left(\frac{b-a}{2}\right) dt \approx \frac{b-a}{2} \sum_{j=1}^n f(\Phi(\hat{c}_j))\,\hat{\omega}_j$$

(stretch)

on $[a,b]$ ← $\omega_j = \frac{1}{2}(b-a)\hat{\omega}_j$, $c_j = \Phi(\hat{c}_j)$ ← on $[-1,1]$

Quadrature by interp & approx

↳ scheme $A: C^0[a,b] \to V :=$ space of simple fcts

$Q_n(f) := \int_a^b (Af)(t)\,dt$

interp. operator $I: \mathbb{R}^n \times \mathbb{R}^n \to V$, $((c_j)_j, (q_j)_j) \to I_T(q)$

$$\int_a^b f(t)\,dt \approx \int_a^b I_T((f(c_j))_j)(t)\,dt \approx \sum_{j=1}^n f(c_j) \underbrace{\int_a^b (I_T(e_j))(t)\,dt}_{\omega_j}$$

error: $E_n(f) := \left| \int_a^b f(t)\,dt - Q_n(f) \right| \le |b-a|\,\|f - Af\|_{L^\infty}$

polyn. quadrature formulas

$I_T :=$ lagrange interp in $c_1, \dots, c_n$ $\Rightarrow Q_n(f) = \sum f(c_j)\omega_j$, $\omega_j := \int_a^b L_{j-1}(t)\,dt$ (↳ $(j-1)$th lagr. polyn)

Newton-Cotes formulas (global)

$n=0$ → 1 point formula

mid-point formula $(b-a) \cdot f(\frac{1}{2}(a+b))$ — $c_1 = \frac{1}{2}(a+b)$, $\omega_1 = b-a$

$n=1$ → **trapezoidal rule** (2 point formula) $\frac{(b-a)}{2}(f(a)+f(b))$ — $c_1 = a$, $c_2 = b$, $\omega_1 = \omega_2 = \frac{b-a}{2}$

$n=2$ → **simpson rule** $\frac{h}{3}(f_0 + 4f_1 + f_2)$

$n=3$ → **milne rule** $\frac{3h}{8}(f_0 + 3f_1 + 3f_2 + f_3)$

$n=4$ → **weddle rule** $\frac{2}{45}h(7f_0 + 32f_1 + 12f_2 + 32f_3 + 7f_4)$

$f_i := f(c_{i,1})$, $h = \frac{b-a}{n}$ ⇒ interp with poly of equally spaced points → equidistant instances for $n \gg 1$ ⇒ don't use!

$n>7$ → weights get negative → compromises numerical stability (use chebychev or clenshaw-curtis)

GAUSS QUADRATURE

quality criterion:

order of quadrature: of $Q_n(f): C^0[a,b] \to \mathbb{R}$ is

$\text{order}(Q_n) := \max\{q \in \mathbb{N}_0 : Q_n(p) = \int_a^b p(t)\,dt \ \ \forall p \in P_q\}+1$ — max deg+1 of polyn for which quadrature is exact

type = better

P_n invariant under **affine pullback** → order not affected by transformation

sufficient order conditions

$Q_n(f)$ has order $\ge n \Leftrightarrow \omega_j = \int_a^b L_{j-1}(t)\,dt$ → proof: $\{L_0, \dots, L_{n-1}\}$ spans P_{n-1}

max order of Q_n is $2n$ — proof: assume order$(Q_n) = 2n+1 \Leftrightarrow$ exact $\forall q \in P_{2n}$

▷ $q(t) = \prod_{j=1}^n (t-c_j)^2 \in P_{2n}$ (all c_j = zeros)

$Q_n(q) = 0$ but $\int_a^b q(t)\,dt > 0$ — q isn't exact ∎

$Q_n(L_j) = \omega_{j+1} = (\text{exact}) \int_a^b L_j(t)\,dt$

ex: 2pt quadratures, orders 4

n-pt rule ⇒ $2n$ free param (ω, c) } enough data.

$\dim P_{2n-1} = 2n$ → $2n$ equations

$Q_n(p) = \int_a^b p(t)\,dt \ \forall p \in P_3 \Leftrightarrow Q_n(\xi^q) = \frac{1}{q+1}(b^{q+1} - a^{q+1})$, $q = 0,1,2,3$

⇒ 4 equations

$$\int_{-1}^1 1\,dt = 2 = 1\omega_1 + 1\omega_2 \qquad \int_{-1}^1 t\,dt = 0 = c_1\omega_1 + c_2\omega_2$$

$$\int_{-1}^1 t^2\,dt = \frac{2}{3} = c_1^2\omega_1 + c_2^2\omega_2 \qquad \int_{-1}^1 t^3\,dt = 0 = c_1^3\omega_1 + c_2^3\omega_2$$

$\Rightarrow \omega_1 = \omega_2 = 1$, $c_1 = 1/\sqrt{3}$, $c_2 = -1/\sqrt{3}$

existence of such order quadrature

- $\{\bar{P}_n\}_{n \in \mathbb{N}_0}$: non-zero polyn → $\bar{P}_n \in P_n$, $\bar{P}_n := \prod_{j=1}^n (t-c_j)$
  - $\int_{-1}^1 q(t)\bar{P}_n(t)\,dt = 0 \ \ \forall q \in P_{n-1}$ (order $2n-1$)
    $= \sum_{j=1}^n \omega_j (q\bar{P}_n)(c_j)$, $\omega_j = \int_a^b L_{j-1}(t)\,dt$
  - $\{c_j\}_{j=1}^n$ are real zeros of $\bar{P}_n$

⇒ $\bar{P}_n$ is unique

Legendre polyn

n^{th} legendre polyn $P_n \in P_n$

$\int_{-1}^1 P_n(t) q(t)\,dt = 0 \ \ \forall q \in P_{n-1}$

$P_n = \frac{1}{2^n} \sum_{k=0}^n \binom{n}{k}^2 (x-1)^{n-k}(x+1)^k$

- $P_n(1) = 1$ (normalization)
- $P_0 = 1$, $P_1 = x$, $P_2 = \frac{1}{2}(3x^2-1)$, $P_3 = \frac{1}{2}(5x^3-3x)$, $P_4 = \frac{1}{8}(35x^4 - 30x^2 + 3)$, $P_5 = \frac{1}{8}(63x^5 - 70x^3 + 15x)$

P_n has n distinct zeros in $]-1,1[$ → they are gauss points

Gauss-Legendre quadrature rules ($\xi_j^n :=$ zeros) → (global)

the weights are all positive

$\omega_i = \frac{2}{(1-x_i^2)(P_n'(x_i))^2}$

→ see proof

→ see proof (compute inverting sign polyn → result doesn't change ±)

computation of nodes

$P_{n+1}(t) = \frac{2n+1}{n+1} t P_n(t) - \frac{n}{n+1} P_{n-1}(t)$

$\tilde{P}_n := \frac{1}{\sqrt{n+\frac{1}{2}}} P_n$ satisfy $\tilde{P}_{-1} = 0$

↳ $t\tilde{P}_n(t) = \frac{n}{\sqrt{4n^2-1}} \tilde{P}_{n-1}(t) + \frac{n+1}{\sqrt{4(n+1)^2-1}} \tilde{P}_{n+1}(t)$ — $:= \beta_{n+1}$

$\beta_n := \frac{n}{\sqrt{4n^2-1}}$ — **golub-welsch**

$$t \begin{bmatrix} \tilde{P}_0(t) \\ \vdots \\ \tilde{P}_{n-1}(t) \end{bmatrix} = \begin{bmatrix} 0 & \beta_1 & & \\ \beta_1 & 0 & \beta_2 & \\ & \beta_2 & \ddots & \\ & & \ddots & \beta_{n-1} \\ & & \beta_{n-1} & 0 \end{bmatrix} \begin{bmatrix} \tilde{P}_0(t) \\ \vdots \\ \tilde{P}_{n-1}(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \beta_n \tilde{P}_n(t) \end{bmatrix}$$

if $\tilde{P}_n(\xi) = 0 \Rightarrow \xi \rho(\xi) = J_n \rho(\xi)$ ↳ EW of J_n

quadrature error ↔ best approx error

Q.R of order $q \ge 1$ → $E_n(f) =$ quadrature error, (by linearity $E_n(f-p) = E_n(f)$)

$E_n(f) \le \|f-p\|_{L^\infty} (|b-a| + \sum_{j=1}^n |\omega_j|)$ — if $\omega_j \ge 0$ use $\sum_{j=1}^n \omega_j = b-a$

$\Rightarrow E_n(f) \le 2|b-a|\,\|f-p\|_{L^\infty}$ $\forall p \in P_{q-1}$

inf: best approx error

convergence analysis:

$f \in C^r \Rightarrow |E_n(f)| = O(n^{-r})$ alg. cvg → $E_n(f) \approx Cn^{-r}$

f analytic ext to $\mathbb{C}$-neighborhood of $[a,b]$ $\Rightarrow |E_n(f)| = O(q^n)$ exp. cvg (faster cvg) ↳ chebychev interp → $E_n(f) \approx Cq^n$

additional effort to reduce quad error by $\rho > 1$

$\frac{Cn_1^{-r}}{Cn_0^{-r}} \approx \frac{1}{\rho} \Rightarrow \frac{n_1}{n_0} \approx \sqrt[r]{\rho}$ — bigger rate r → less additional effort

$q^{n_1 - n_0} = \frac{1}{\rho} \Rightarrow n_1 = n_0 - \frac{\log\rho}{\log q}$

smoothness of integrand is essential for fast cvg

smooth by transformation

ex $\int_a^b \sqrt{t}\,f(t)\,dt \to \int_0^{\sqrt{b}} s f(s^2) 2s\,ds \to \int_a^b \frac{2\tau^2}{b} f\left(\frac{\tau^2}{b}\right) \frac{1}{\sqrt{b}}\,d\tau$

$s := \sqrt{t}$, $\tau = \sqrt{b}\,s$

↳ no smooth → slow cvg; ↳ exp cvg (has analytic); → the cvg is now exponential. original $\in C^0$, now $\in C^\infty$ (no changes like before) → gauss ensures exp if smooth integrand ($\in C^\infty$)

COMPOSITE QUADRATURE

$M := \{a = x_0 < x_1 < \dots < x_m = b\}$

$\int_a^b f(t)\,dt = \sum_{\ell=1}^m \int_{x_{\ell-1}}^{x_\ell} f(t)\,dt$ → apply quadrature here locally

#f-eval = $\sum_{\ell=1}^m (n_\ell)$ → local n_ℓ-point quadrature

composite trapezoidal $\int_a^b f(t)\,dt \approx \frac{1}{2}(x_1 - x_0) f(a) + \sum_{j=1}^{m-1} \frac{1}{2}(x_{j+1} - x_{j-1}) f(x_j) + \frac{1}{2}(x_m - x_{m-1}) f(b)$

composite simpson $\int_a^b f(t)\,dt \approx \frac{1}{6}(x_1 - x_0) f(a) + \sum_{j=1}^{m-1} \frac{1}{6}(x_{j+1} - x_{j-1}) f(x_j) + \sum_{j=1}^m \frac{2}{3}(x_j - x_{j-1}) f\left(\frac{1}{2}(x_j + x_{j-1})\right) + \frac{1}{6}(x_m - x_{m-1}) f(b)$

special: all local quad rules from a single qr on reference interval by affine transf.

error estimates: add together local error contributions

$f \in C^r[a,b] \to \left| \int_{x_{j-1}}^{x_j} f(t)\,dt - Q_{n_j}^j(f) \right| \le C |x_j - x_{j-1}|^{\min\{r, q_j+1\}} \left\| f^{(\min\{r, q_j\})} \right\|_{L^\infty}$ ↳ local order

if $q_j = q \ \forall j$ $\left| \int_{x_0}^{x_m} f(t)\,dt - Q_n(f) \right| \le C h_M^{\min\{q,r\}} |b-a| \left\| f^{(\min\{q,r\})} \right\|_{L^\infty}$ ↳ max meshwidth $|x_j - x_{j-1}|$

n_j the same for all j $\Rightarrow E_n(f) = O\left(n^{-\min\{q,r\}}\right) = O\left(h_M^{\min\{q,r\}}\right)$ for $n = \#\text{f-eval} \to \infty$

h-convergence

comparison: composite quad vs G-L quad

$E_n^{CR}(f) \le Cn^{-\min\{q,r\}}$ — $E_n^{GL}(f) \le Cn^{-r}$ (at least as good as CR)

if f analytic, GL has exp. rate $\Rightarrow$ WINNER ↳ here it's also composite

$\phi: [0,1[\to S^1 = \{z \in \mathbb{C} : |z| = 1\}$

quadrature on S: $Q_n^S(g) = \sum_{j=1}^n w_j^S g(z_j^S) = \int_S g(z)\,ds(z)$

→ equidistant lagrange interp.

nodes $z_j := \exp(2\pi i\, j/n)$

polyn. interp with complex nodes

→ same theory. weights (perfect symmetry of nodes) $w_j^S = \frac{2\pi}{n}$

$$\int_0^1 f(t)\,dt = \frac{1}{2\pi}\int_S ((\phi^{-1})^* f)(z)\,dz \approx \frac{1}{2\pi}\frac{2\pi}{n}\sum_{j=0}^{n-1}((\phi^{-1})^* f)(\exp(2\pi i\, j/n)) = \frac{1}{n}\sum_{j=0}^{n-1} f(\phi^{-1}(\phi(j/n))) = \frac{1}{n}\sum_{j=1}^{n-1} f(j/n)$$

(trapezoidal rule)

equidistant T.R ⇒ global polyn. quadr on S^1

error ∼ approx error of equidistant lagr. interp on S

Exp. cvg if $(\phi^{-1})^* f$ analytic ext.

ADAPTIVE QUADRATURE

rehabilitation of composite quadrature (no sense on uniform mesh, make it dependent on f)

use trapezoidal $\left|\int_{x_{k-1}}^{x_k} f(t)\,dt - \frac{h_k}{2}(f(x_{k-1}) + f(x_k))\right| \le \frac{1}{2}\|f''\|_{L^\infty} \cdot h_k^3$ ($:= B_k$)

density (smaller cells on critical pts)

Bound for error: $E(f) \le \sum_k B_k h_k^3$

goal: minimize this under constraint $\sum_k h_k = b-a$

focus on 2 cells: minimize their combined error contribution

$B_k(h_k+\delta)^3 + B_\ell(h_\ell-\delta)^3 \to \min$ (find such δ) → minimizer δ^*

optimal cell size distrib: all cells make same error contribution $B_k(h_k+\delta^*) = B_\ell(h_\ell-\delta^*)$

(error equidistribution principle) → mesh for posteriori adaptive comp. quad adapted to f (depends on f)

nodes selection: a_j^n based on f

priori adapt. quad: nodes are fixed before evaluation (take into account info about f)

posteriori: nodes position selected/improved based on info collected during a loop. terminates when sufficient accuracy has been reached

iterative improvement: move nodes / add nodes (refinement)

Adaptation loop:

- ESTIMATE: based on available info compute approx error on mesh intervals
- CHECK TERM: if tot error small enough → STOP
- MARK: single out mesh intervals with larges/above average errors contributions
- REFINE: add node(s) inside marked intervals

ESTIMATE; idea: local error estimate by comparing local result of 2 quadr. formula of diff order Q_1, Q_2

error $(Q_2) \ll$ error $(Q_1) \Rightarrow$ error $(Q_1) \cong Q_2(I) - Q_1(I)$ | ex $Q_1 \to$ trap. rule (order 2), $Q_2 \to$ simp. rule (4)

error total is crude upper bound for quadrature error

MARK: $S := \{k \in \{1,\dots,m\} : \text{est}_k \ge \eta \cdot \frac{1}{m}\sum_{j=1}^m \text{est}_j\}$, $\eta \cong 0.9$ (what is wished)

REFINE new mesh $M^* = M \cup \{p_k := \frac{1}{2}(x_{k-1}+x_k) : k \in S\}$ % split intervals with too much error

extra: $\int_{-\infty}^{\infty} f(t)\,dt = \int_0^{\pi} \frac{f(\cot(s))}{\sin^2 s}\,ds$ (substitution $t = \cot(s)$)

$F(x) = \arctan(x)$, $DF(x) = \frac{1}{1+x^2}$

example: $f(x) = A*x + C.*\exp(x)$, $DF(x) = A + \text{diag}(C.*\exp(x))$

11. NUMERICAL INTEGRATION

The numerical solution of **ODEs** (ordinary differential equations)

$y' = f(t) \Rightarrow y(t) = \int_0^t f(\tau)\,d\tau$

IVP (initial value problem for ODE's)

1st order ODE in standard form: $\dot y(t) = f(t, y(t))$ $f: I \times D \to \mathbb{R}^d$, $I \subset \mathbb{R}$, $D \subset \mathbb{R}^d$

f = **rhs** (right hand side)

t := time variable

$y(t)$:= state $\in \mathbb{R}^d$, $D \subset \mathbb{R}^d$ = state space. $\dot y(t) = \frac{d}{dt}y(t)$ | $y: I \subset \mathbb{R} \to D$

$$\begin{bmatrix}\dot y_1(t)\\ \dot y_2(t)\end{bmatrix} = \begin{bmatrix} f_1(t, y_1(t), \dots, y_d(t)) \\ f_d(t, y_1(t), \dots y_d(t))\end{bmatrix}$$

Assume f continuous

Solution: $y: J \subset I \to D$, with $\dot y(t) = f(t, y(t))\ \forall t \in J$ ↳ "function of time"

f smooth in $t, y \Rightarrow y$ smooth

Smoothness: ▷ y solution of ODE If $f \in C^r$ (r times continuously diff) with respect to both args, then $t \to y(t) \in C^{r+1}$

ex: population dynamics (growth of pop with limited resources)

$\dot y = (\lambda - c)y = f(y)$ logistic ODE (indep. of time), $d=1$

y = density of pop. $D = \mathbb{R}^+$

$y(t) = \frac{y_0}{y_0 + (1-y_0)e^{-t}}$ | $y_0 = y(0)$

(λ param family of functions, 1 condition y_0 needed to have unique sol)

autonomous = logistic ODE

rhs doesn't depend on t, $\dot y = f(y)$ → always use $t_0 = 0$ ($y(t)$ sol ⇒ $y(t-\tau)$ sol)

lotka - volterra ODE (predata/preq model)

$d=2$ (2 eqns)

preq: $\dot u = (\alpha - \beta v)u$

predo: $\dot v = (\delta u - \gamma)v$

$\alpha\beta\gamma\delta$ parameters representing species interactions

$\iff y = f(y)$, $y = \begin{bmatrix}u\\v\end{bmatrix}$, $f(y) = \begin{bmatrix}(\alpha-\beta v)u\\(\delta u-\gamma)v\end{bmatrix}$

Vector field A solution (= curve in state space) $t \to y(t)$ is the path of a floating particle in velocity field f

stationary point (stable) — periodic solutions

$y(t) = y^*$, $f(y^*) = 0$

ex: transient circuit modeling

Kirchoff's law $i_R(t) - i_C(t) - i_L(t) = 0 \Rightarrow \frac{d}{dt}i_R - \frac{d}{dt}i_C - \frac{d}{dt}i_L = 0$

$i_R(t) = R^{-1}U_R(t)$, $i_C(t) = C\frac{d}{dt}U_C(t)$, $U_L(t) = L\frac{d}{dt}i_L(t)$

$\frac{1}{R}\frac{d}{dt}U_R - C\frac{d^2}{dt^2}U_C - \frac{1}{L}U_L = 0 \iff$ introduce time dependent nodal potentials $\frac{1}{R}\frac{d}{dt}(U_S(t) - U(t)) - C\frac{d^2}{dt^2}U(t) - \frac{1}{L}U(t) = 0$ | 2nd order ODE

theory of IVP

$y' = y \Rightarrow y(t) = C e^t, C \in \mathbb{R}$ (1 param. family) → need to specify initial val

IVP: $y' = f(t,y)$, $y(t_0) = y_0$

Autonomization by introducing t as additional state component

$\dot y = f(t,y)$: $z(t) = \begin{bmatrix}y(t)\\ t\end{bmatrix}$ solves $z' = \begin{bmatrix}f(z_{d+1}, z_1, \dots, z_d)\\ 1\end{bmatrix}$ → equivalent autonomous ODE ($z_{d+1} = t$)

Conversion to 1st order ODE

$\ddot y = f(y, \dot y)$ $z(t) = \begin{bmatrix}y(t)\\ \dot y(t)\end{bmatrix}$ solves $z' = \begin{bmatrix}z_{d+1}, \dots, z_{2d}\\ f(z_1, \dots, z_d, z_{d+1}, \dots, z_{2d})\end{bmatrix}^T$ → equivalent 1st order ODE

for 2nd order ODE: IVP requires values of $y(t_0)$, $y'(t_0)$

Peano & Picard-Lindelöf

If rhs function f is locally Lipschitz continuous ($y \to f(t,y)$ has finite slope around every point)

$\exists L > 0: \|f(t,\omega) - f(t,z)\| \le L\|\omega - z\|$, then for all initial conditions (t_0, y_0) the IVP has 1 solution unique

$y \in C^1$ with maximal (temporal) domain of definition $J(t_0, y_0) \subset \mathbb{R}$

ex: not locally L.C: $t \to \sqrt{t}$ on $[0,1]$

ex: blowup $y' = y^2$ → $y(t) = \begin{cases}\frac{1}{y_0^{-1} - t} & \text{else} \\ 0 & \text{if } y_0 = 0\end{cases}$ domains of def: $J(y_0) = \begin{cases}]-\infty, y_0^{-1}[& y_0 > 0 \\ \mathbb{R} & y_0 = 0 \\ ]y_0^{-1}, \infty[& y_0 < 0\end{cases}$

evolution operators

assume global solution $J(0, y_0) = \mathbb{R}$

evolution map: $\Phi: \begin{cases}\mathbb{R} \times D \to D \\ (t, y_0) \to y(t, y_0) = \Phi^t y_0\end{cases}$ y_0 fixed: → trajectories; t fixed: → state mapping

where we would find each other at same time, different y_0

$\frac{\partial \Phi}{\partial t}(t, y) = f(\Phi(t,y))$, $f(z) = \frac{\partial \Phi}{\partial t}(1, (\Phi^t)^{-1}(z))$

polynomial approx. methods

we want $y(T)$ for some "final time" T, and approximate trajectory $t \to y_N(t) \cong y(t)$

temporal mesh: $M = \{t_0 < t_1 < \dots < t_N = T\}$

explicit euler method: follow tangents over short times ($f(t, y(t))$ gives slope in t)

$y_k \cong y(t_k)$

1st step. then repeat with y_1 (timestepping) → produce sequence $y_0, y_1, \dots, y_N$ with pw lin interp → polygon

(forward)

recursion formula: $y_{k+1} = y_k + (t_{k+1} - t_k) f(t_k, y_k)$ ($y_k'(t_k)$)

difference quotient $\frac{y_{k+1} - y_k}{t_{k+1} - t_k} = f(t_k, y_k)$

implicit euler method

choose anchor point t_{k+1} (use tangent in next point)

use $f(t_{k+1}, y_{k+1})$

$= f(t_{k+1}, y_{k+1})$

solve for $y_{k+1} = y_k + h_k f(t_{k+1}, y_{k+1})$. y_{k+1} solves $G(h_k, y_{k+1}) = 0$, $G(h,y) = y - y_k - hf(y)$

$h_k := t_{k+1} - t_k$

$G(0, y_k) = 0$

$D_2 G(0,y) = I$ invertible for small h, $G(h, y_k) = 0$ defines $h \to y(h)$ | y_{k+1} exists for sufficient small h

implicit midpoint method

diff. quot $\cong f\left(\frac{t_{k+1} + t_k}{2}, \frac{y_{k+1} + y_k}{2}\right)$ → polygonal approximation

(tendence on middle of 2 pts)

$y_{k+1} = y_k + h_k f(\cdot\cdot)$. All polygonal methods (for sufficient small h_k) define recursion

timestepping sequence $y_0, y_1, \dots, y_N$, hope $y_k \cong y(t_k)$

$y_{k+1} = \Psi(h_k, y_k)$

$y(t_{k+1}) = \Phi(h_k, y(t_k))$

Ψ's or approximation, discrete evolution

GENERAL SINGLE STEP METHODS (SSM)

goal: $\Psi \cong \Phi$

▷ given Ψ, y_0, M, the recursion $y_{k+1} = \Psi(h_k, y_k)$ is a SSM for autonomous IVP $\dot y = f(y)$, $y(0) = y_0$

$y_{k+1} = \Psi^{h_k} y_k$ Ψ is of the form $\Psi^h y = y + h\psi(h,y)$, $\psi: \begin{cases} I \times D \to \mathbb{R}^d \text{ continuous} \\ \psi(0,y) = f(y)\end{cases}$

consistent

convergence:

Assume f lipschitz continuous

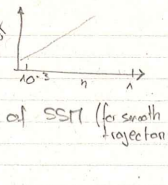
error: $\max_j \|y_j - y(t_j)\|$

Need family of meshes $(M_\ell)_\ell$: $h_\ell := \max_j |t_{j+1}^\ell - t_j^\ell| \to 0$ as $\ell \to \infty$

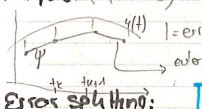
h-refinement

meshwidth

ex: $q' = \lambda q(1-q)$, $q_0 = 0{,}01$, final time $T=1$, $h = \frac{1}{N}$, $N = \#$ IH
⇒ algebraic avg for $h \to 0$
euler: $\mathcal{O}(h)$ (first order)
imp. method: $\mathcal{O}(h^2)$
→ In general, consistent SSM converge algebraically for $h \to 0$
error $= \mathcal{O}(h^p)$, $p \in \mathbb{N}$ order of SSM (for smooth trajectories)



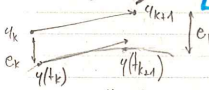
Analysis: (expl. euler)



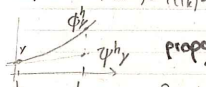
$|$ = error sequence $e_k = q_k - q(t_k)$
$\mathcal{Y}$ = discrete evolution between points

Error splitting:

$$e_{k+1} = \mathcal{Y}^{hk} q_k - \phi^{hk} q(t_k) = \underbrace{\left(\mathcal{Y}^{hk} q_k - \mathcal{Y}^{hk} q(t_k)\right)}_{\text{propagated error (from before)}} + \underbrace{\left(\mathcal{Y}^{hk} y(t_k) - \phi^{hk} q(t_k)\right)}_{\text{new error}}$$



Taylor: $q(t_{k+1}) = q(t_k) + q'(t_k)h + \frac{1}{2}h^2 q''(\xi) = \mathcal{Y}^h y(t_k) + \frac{h^2}{2} q''(\xi)$, $\xi \in]t_k, t_{k+1}[$



propagated error: $= \|e_k + h(f(y(t_k)) - f(q_k))\| \leq \|e_k\| + hL\|e_k\|$ (Lipschitz-cont)

$e_{k+1} \leq \underbrace{(1+hL)}_{\text{propagated}} e_k + \underbrace{\tau_k}_{\text{one step error}}$, $\tau_k = \frac{1}{2}h^2 q''(\xi_k)$, $e_0 = 0$

[...] $(1+x \leq e^x)$ $e_k \leq e^{TL} \|q''\|_{L^\infty[0,T]} hT$ $\mathcal{O}(h)$

Note: one step error $\mathcal{O}(h^2)$ ↔ tot error $\mathcal{O}(h)$
for SSM: — $\mathcal{O}(h^{p+1})$ ↔ order p

EXPLICIT RUNGE-KUTTA SSM

goal: explicit SSM with order $p > 1$
Fehlende for high orders:
Discretization error $\cong$ Ch^p [Cunkraun, assume sharp]
goal: error reduction by factor $\rho > 1$ using h-refinement
effort $\sim$ #{timesteps} $\sim h^{-1}$

$\frac{Ch_0^p}{Ch_1^p} = \rho \Rightarrow h_1 = \rho^{-1/p} \cdot h_0$
$\omega_1 = \rho^{1/p}(\omega_0)$ (effort after refinement) ↔ (effort before)

less additional effort to achieve prescribed error reduction the larger p

$q' = f(t,q) \Rightarrow q(h) = q_0 + \int_0^h f(q(t))dt$
$q(0) = q_0$ $\quad q_1 = q_0 + h\sum_{i=1}^s b_i \cdot f(q(c_i h))$ ⇒ **idea: quadrature** (approximate integral) — s-point quadrature on $[0,1]$, weights b_i, nodes c_i

bootstrapping $q(1.h) \cong q_1$ → used to get these values

goal: one-step error $\mathcal{O}(h^{p+1})$ → sufficient to approximate $q(t_0 + c_i h)$ with error $\mathcal{O}(h^p)$, because of multiplication with h!

ex: quad formula = trapezoidal rule $Q(f) = \frac{1}{2}(f_0 + f_1) \to s=2$, $c_1=0$, $c_2=1$, $b_1 = b_2 = \frac{1}{2}$

$q(t_1) \cong q_1 = q_0 + \frac{h}{2}(k_1 + k_2)$, $k_1 = f(t_0, q_0)$, $k_2 = f(t_0 + h, q_0 + hk_1)$ — approximated by explicit euler step

midpoint rule: $+ hk_2$, $k_2 = f(t_0 + \frac{h}{2}, q_0 + \frac{h}{2}k_1)$

def explicit RK-SSM

for $b_i, a_{ij} \in \mathbb{R}$, $c_i = \sum_{j=1}^{i-1} a_{ij}$, $i,j = 1,\dots,s \in \mathbb{N}$, an s-stage expl. RK-SSM for ODE $\dot{q} = f(t,q)$ is defined as

$$k_i = f(t_0 + ch, q_0 + h\sum_{j=1}^{i-1} a_{ij} k_j), \quad i = 1..s, \qquad q_1 := q_0 + h\sum_{i=1}^s b_i k_i$$

$q_{n+1} = q_n + \dots$ $\quad c_1 = 0$, $c_2 = a_{21}$, $c_3 = a_{31} + a_{32}$, $c_4 = a_{41} + a_{42} + a_{43}$

$t_{j+1} = t_j + h$

vectors k_i are called **increments**
$h > 0$ is the size of the timestep

effort per step: s f-eval

Butcher scheme (shorthand notation)

$$\begin{array}{c|c} c & A \text{ (strictly lower triangular)} \\ \hline & b^T \end{array} = \begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1s} \\ \vdots & \vdots & & \vdots \\ c_s & a_{s1} & \cdots & a_{ss} \\ \hline & b_1 & \cdots & b_s \end{array}$$

consistent ⇔ $\sum_{i=1}^s b_i = 1$

$\begin{array}{c\|c} 0 & 0 \\ \hline & 1 \end{array}$	$\begin{array}{c\|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$	$\begin{array}{c\|cc} 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ \hline & 0 & 1 \end{array}$	$\begin{array}{c\|ccc} 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 1 & 0 & 0 & 1 \\ \hline & \frac{1}{6} & \frac{2}{6} & \frac{2}{6} & \frac{1}{6} \end{array}$
expl. euler method, order = stages = 1	implicit euler method, =1	expl. trapezoid rule, =2	expl. midpoint rule, =2	Classical 4th order RK-SSM

Construction high-order RK-SSM by solving order conditions

order p	1	2	3	4	5	6	7	8	≥9
min. # of stages s	1	2	3	4	6	7	9	M	$\geq p+3$

... theory: $p \leq s$, $p < s$ (if $p \geq 5$)

MATLAB: $[t,q] = \text{ode45}(\text{odefun}, \text{tspan}, q_0, \text{options})$

t: temporal mesh $\{t_0 < t_1 \dots < t_{N-1} = T\}$
q: column vectors

handle to function $\dot{q} = f(t,q)$ ↔ ch.s $f(t,q)$
tspan: vector $(t_0, T)^T$ with start/end time
q_0: vector with initial state

Adaptive stepsize control

example: oscillatory chemical reaction (BZ-reaction)
↳ es: in the case of blowup it detects it and solves the problem

Objective: N as small as possible & $\max_{k=1\dots N} \|q(t_k) - q_k\| < TOL$

policy: Try to curb/balance one-step error as $\|q(T) - q_N\| < TOL$
- adjusting current h_k
- predict suitable next timestep h_{k+1}

} local in time stepsize control (choose h_k based on h_{k-1}, q_{k-1} alone)

I. estimate one-step error
→ compare results for 2 discrete evolutions $\mathcal{Y}^h, \tilde{\mathcal{Y}}^h$ of different order ($\tilde{\mathcal{Y}}$ > order $\mathcal{Y}$)

then $\phi^h q(t_k) - \mathcal{Y}^h q(t_k) \cong EST_k := \tilde{\mathcal{Y}}^h q(t_k) - \mathcal{Y}_q(t_k)$

II. refine $EST_k < \max(abstol, reltol \cdot \|q_k\|)$?
↳ has almost nothing to do with $q(t_k) - q_k$
→ accept: go to next step, $h_{k+1} = \alpha h_k$, $\alpha > 1$
reject: repeat with $h_k \leftarrow \frac{1}{2}h_k$
→ abs/rel tol imposed have nothing to do with global error of solution, only one step error

- **Stepsize prediction**

when $EST_k > TOL$ stepsize adjustment (better h_k)
$<$ prediction (good h_{k+1})

$EST_k \cong Ch^{p+1}$ for small h
goal: efficiency $EST = TOL$, h_{new}: $Ch_{new}^{p+1} = TOL$ ⇒ $h_{new} = h \cdot \sqrt[p+1]{\frac{TOL}{EST}}$

$C \cong \frac{EST}{h^{p+1}}$

↳ recommended new timestep

⚠ safeguard against oscillating timesteps

implementation := embedded RK methods (save a_{ij}, different b_i for different orders)

RATE: $\log(err)/\log(h) \to \mathcal{O}(h^{\text{rate}})$

12. SSM FOR STIFF IVP

what's wrong with ode45 and its RK-SSM? (could be oscillations due to overshooting)

Model problem analysis
es $q' = \lambda q$, explicit euler → $q_{k+1} = q_k + h\lambda q_k \Rightarrow q_k = (1+h\lambda)^k q_0$

Adaptive timestepping
↳ detects timestep constraints, small timestep throughout is inefficient
es: explicit trapezoidal method (blow up if >1)
$q' = \lambda q \Rightarrow q_1 = (1 + h\lambda + \frac{1}{2}(h\lambda)^2)q_0 =: S(h\lambda)q_0 \Rightarrow q_k = S(h\lambda)^k q_0$
↳ no blowup if $|\cdot| \leq 1$

for general explicit RK-SSM

$$\begin{bmatrix} I - zA & 0 \\ -zb^T & 1 \end{bmatrix} \begin{bmatrix} k \\ Y_1 \end{bmatrix} = q_0 \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad K = \begin{bmatrix} k_1 \\ \vdots \\ k_s \end{bmatrix}$$

in our case $z = \lambda h$

$\Rightarrow q_1 = q_0 + zb^T(I - zA)^{-1}\mathbb{1} q_0 = S(z)q_0$

Stability function of explicit RK-SSM

the discrete evolution $\mathcal{Y}^h$ of explicit s-stage RKSSM for ode $\dot{q} = \lambda q$ is a multiplication with number $\mathcal{Y}_\lambda^h = S(\lambda h) \Leftrightarrow q_1 = S(h)q_0$

where S is the stability function

$$S(z) = 1 + zb^T(I - zA)^{-1}\mathbb{1} = \frac{\det(I - zA + z\mathbb{1}b^T)}{\det(I - zA)}$$ from cramer's rule

$q_k = S(z)^k q_0$ (uniform timestep $h > 0$)

↳ $\det(I - zA)$ in general RK-SSM $1/3$

- explicit euler: $S(z) = 1 + z$
- expl. trapez. method: $S(z) = 1 + z + \frac{1}{2}z^2$
- classic RK4: $S(z) = 1 + z + \frac{1}{2}z^2 + \frac{1}{6}z^3 + \frac{1}{24}z^4$ $\left(\sum_{i}^{\cdot} \frac{z^i}{i!}\right)$

$e^z = 1 + z + \frac{1}{2}z^2 + \frac{1}{6}z^3 + \dots$
implicit euler: $S(z) = \frac{1}{1-z}$

polynomial stability funct.
S = non const polyn of deg $\leq s$: $S \in \mathbb{P}_s$
↳ $|S(z)| \to \infty$ as $|z| \to \infty$

If you ensure $|\lambda h|$ small enough, you can avoid exponential approximations q_k (qualitatively wrong for $\lambda < 0$) when applying explicit RKSSM

Model problem analysis for linear syst. ODE

$\dot{q} = Mq$, $M \in \mathbb{R}^{d,d}$ solved by

diagonalization $V^{-1}MV = D = \text{diag}(\lambda_1, \dots, \lambda_d)$, $\lambda_i \in \mathbb{C}$ eigenvalues $\in \sigma(M)$ → see linalg
$z = V^{-1}q \Rightarrow z' = Dz$ (decoupled scalar linear ODEs)

expl EU meth for $\dot{q} = Mq$: diagonalization

$y_{k+1} = q_k + hMy_k$ $\quad z_k = V^{-1}q_k \Rightarrow z_{k+1} = z_k + hV^{-1}MV(V^{-1}q_k) = (I + hD)z_k$

$\Rightarrow (z_{k+1})_i = (1 + h\lambda_i)(z_k)_i$ (decoupled)

to avoid blowup ensure $|1 + h\lambda_i| \leq 1 \ \forall i$ $\quad (\lambda_i \in \mathbb{C} \to h|\lambda_i| \leq 2)$

General: $q' = Mq$ $\quad f(q) = Mq$ $\quad \hat{k}_i = V^{-1}k_i = D(z_0 + h\sum_j a_{ij}\hat{k}_j)$

$z_i := z_0 + h\sum_{j=1}^{i-1} a_{ij}\hat{k}_j$

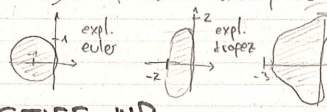
increment equation: $(\hat{k}_i)_\ell = \lambda_\ell\left[(z_0)_\ell + h\sum a_{ij}(\hat{k}_j)_\ell\right]$

generation of uniformly bounded solution sequences $(q_k) \Leftrightarrow$ it generates uniformly bounded sequences for all the scalar ODE's $\dot{z} = \lambda_i z$

$q' = f(q) = Aq$
$\Rightarrow Df(q) = A$

stability: this sequence decays exponentially $\forall q_0 \Leftrightarrow |S(\lambda_i h)| < 1 \ \forall \in \sigma \lambda_i$

Region of absolute stability
discrete ev $\mathcal{Y}$ for $q' = \lambda q$ be $\mathcal{Y}^h q = S(z)q$, $z = h\lambda$.
the region of absolute stability is $S_{\mathcal{Y}} := \{z \in \mathbb{C} : |S(z)| < 1\} \subset \mathbb{C}$ ($\lim_{|z| \to \infty} S(z) = 0$)
↳ if bounded ⚠ blowup!



⚠ $0 \in \partial S_{\mathcal{Y}}$, because $S(z) = 1 + z + \mathcal{O}(|z|^2)$

STIFF IVP

example: chemical reaction → non linear ODE involving concentrations $q(t) = \begin{bmatrix} c_A(t) \\ \vdots \\ c_D(t) \end{bmatrix}$
⚠ $\text{Re}(\lambda)$ in $q(t) = e^{\lambda t}$ governs decay/growth
$\text{Re}\,\lambda_j \leq 0 \ \forall j$ and $|\text{Re}\,\lambda_\ell| \gg 1$
→ exact solution remains bounded
→ blow-up of numerical solution disastrous
→ Avoid blow-up

es: strongly attractive limit cycle ODE $\dot{q} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} q + \lambda(1 - \|q\|^2)q$

NP is called STIFF, if stability imposes much tighter timestep constraints on SSM than the accuracy requirements

heuristic consideration for predicting stiffness
linear ODE $\dot{y} = My$ $\quad$ stiffness state dependant $\quad$ (stiff at state $q^* \in q(I)$?
general ODE $\dot{q} = f(q)$

Linearization $f(q) \cong f(q^*) + Df(q^*)(q - q^*)$
→ close to q^* solutions of $\dot{q} = f(q)$ will behave like solutions of the linearized ODE

$\dot{z} = f(q^*) + Df(q^*)(z - q^*)$ → affine linear ODE

of RK SSM $y_0 = q^*$: $k_i = f(q^*) + Df(q^*)h\sum_{j=1}^{i-1} a_{ij}k_j$ (increment eq for RKSSM applied)

for small timestep the behaviour of RKSSM applied to $\dot{q} = f(q)$ close to q^* is determined by EW of Jacobian $Df(q^*)$

how to distinguish stiffness
NP for autonomous ODE $\dot{q} = f(q)$ will probably be stiff, if for substantial periods of time $\min_\lambda \{\text{Re}(\lambda) : \lambda \in \sigma(Df(q(t)))\} \ll 0$ ($\max_\lambda \{\dots\} \approx 0$)

$\sigma(M)$ is **spectrum of matrix** (set of EW) → see example

IMPLICIT RKSSM

implicit euler method for stiff NP $\quad q_{k+1} = q_k + hf(q_{k+1})$ → apply to $\dot{q} = \lambda q$

$\to q_{k+1} = \frac{1}{1-h\lambda} q_k = \left(\frac{1}{1-h\lambda}\right)^k q_0$ $\quad$ if $\lambda < 0 \Rightarrow \left|\frac{1}{1-h\lambda}\right| < 1 \Rightarrow \lim_{k\to\infty} q_k = 0$ $\quad (\forall h > 0)$

No stability induced timestep constraint
Same result for linear ODEs: $\dot{q} = Mq$

for any timestep, the impl. EU method generates exponentially decaying solution sequences for $\dot{q} = Mq$ with EW $\lambda_1, \dots, \lambda_d$ if $\text{Re}\,\lambda_i < 0 \ \forall i$

COLLOCATION SSM

first step of SSM for IVP: $\dot q = f(q)$, $q(0)=q_0$

→ q_1, stepsize

1. On $[0,h]$ approximate $q(t) \approx q_n(t) \in V :=$ finite dim space of functions $[0,h] \to \mathbb{R}^d$. standard choice $V = (\mathcal{P}_s)^d$, $\dim V = s+1$
2. selection of q_n through collocation conditions $\begin{cases} q_n(0) = q_0, \\ \dot q_n(\tau_j) = f(q_n(\tau_j)) \end{cases}$ for collocation points $0 \le \tau_1 \in \dots \le \tau_s = h$
   → $s+1$ equations
3. $q_1 = q_n(h)$

collocations points from reference interval $[0,1]$: $\tau_j = c_j h$ $\quad 0 \le c_1 \le c_2 \le \dots \le c_s = 1$

$q_n \in \mathcal{P}^d_{s-1} \Rightarrow \dot q_n(\tau h) = \sum_{j=1}^s \dot q_n(hc_j) L_j(\tau)$ $\quad \tau \in [0,1]$

$\{L_j\}_{j=1}^s$ = lagrange polynomials of degree $s-1$ for node set $\{c_j\}_{j=1}^s$: $L_j(c_i) = \delta_{ij}$

$\Rightarrow \dot q_n(\tau h) = \sum_{j=1}^s f(q_n(c_j h)) L_j(\tau)$ $\quad q_n(\xi h) - q_n(0) = h \sum_{j=1}^s f(q_n(c_j h)) \int_0^\xi L_j(\tau)\,d\tau$

→ same as RK-SSM $\quad a_{ij} = \int_0^{c_i} L_j(\tau)\,d\tau$ $\quad b_i = \int_0^1 L_i(\tau)\,d\tau$ $\quad =: k_j$

collocation ssm (CL-SSM)

$\Rightarrow$ k_i: to be determined by solving a system of equations (→ implicit

CL-SSM applied to $\dot q = f(t)$, $q(0)=0$ $\hat{=}$ a polynomial quadrature formulas $(k_i = f(c_i h))$

convergence:

ex $q' = 10q(1-q)$, $q_0 = 0{,}01$, $\tau = 1$ $\quad$ equidistant collocation points $c_j = \frac{j}{s+1}$, $j = 1 \dots s$

algebraic ord, explicit rules

$s=1$	$p = 1{,}96$	$1{,}96$
2	2,03	4,01
3	4,00	6,00
4	4,004	8,02

} Gauss order $2s$ = order of s-pt Gauss quadrature

order of collocation SSM

$f \in C^p(I \times D)$, the order of an s-stage collocation SSM agrees with the order of the quadrature formula on $[0,1]$ with nodes c_j and weights b_j

General RK-SSM

apply following modifications to explicit RK-SSM:

$c_i := \sum_{j=1}^s a_{ij}$ $\quad k_i = f(t_0 + c_i h, q_0 + h \sum_{j=1}^s a_{ij} k_j)$

→ Butcher Scheme $\begin{array}{c|c} c & A \\ \hline & b^T \end{array}$ is the same, only A can now be sxs (not upper triang)

k_i: obtained by solving increment equations iteratively (simplified newton)

↳ freeze Jacobian

example: $g_i := h \sum_{j=1}^s a_{ij} k_j = h \sum_{j=1}^s a_{ij} f(q_0 + g_j) \quad \Leftrightarrow k_i = f(q_0 + g_i)$

$\vec g = \begin{bmatrix} g_1 \\ g_2 \end{bmatrix} \in \mathbb{R}^2 : \quad F(\vec g) = 0 = \vec g - hA \begin{bmatrix} f(q_0+g_1) \\ f(q_0+g_2) \end{bmatrix}$

Initial guess for simplified newton: (h small) $\quad \vec g^{(0)} = 0$

$DF(0) = I - hA \begin{bmatrix} \frac{\partial f}{\partial q}(q_0) & & 0 \\ & \ddots & \\ 0 & & \frac{\partial f}{\partial q}(q_0) \end{bmatrix}$

iteration: $\vec g^{(i+1)} = \vec g^{(i)} - DF(0)^{-1} F(\vec g^{(i)})$ $\quad S = I - h \frac{\partial f}{\partial q}(q_0) A$ $\quad \frac{\partial f}{\partial q}(q_0)$

↳ invertible for h small enough

Recap example: implicit euler: $S_{SM} := \{z \in \mathbb{C} : |z-1| > 1\}$

implicit midpoint rule

$\begin{array}{c|c} 1/2 & 1/2 \\ \hline & 1 \end{array}$ $(s=1) \Rightarrow S(z) = \frac{1 + \frac{1}{2}z}{1 - \frac{1}{2}z}$, $S_{SM} = \mathbb{C}^-$ (unconditional stability, ideal region of stability)

$\mathrm{Re}(\lambda) < 0 \Rightarrow h\lambda \in S_{SM} \Rightarrow$ decaying q_k

$> 0 \Rightarrow |S(h\lambda)| > 1 \Rightarrow$ growing q_k (exact motch for growth/decay of exact solution $q(t) = Ce^{\lambda t}$)

- **Region of stability of Gauss-collocation**

s-stage Gauss collocation SSM, nodes c_s given by s gauss points on $[0,1]$

features ideal stability domain $S_{SM} = \mathbb{C}^-$

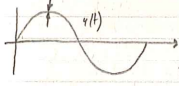
in particular, they all are A-stable

A-stable: unconditionally stable for decay eq.

Def: RKSSM with stability function is A-stable, if $\mathbb{C}^- := \{z \in \mathbb{C} : \mathrm{Re}(z) < 0\} \subset S_{SM}$

(won't be affected by stability induced timestep constraint on stiff IVP)

the catch: example $\dot q = -\lambda q + \beta \sin(2\pi t)$, $\lambda = 10^6$, $\beta = 10^6$, $q(0) = 1$



$t \to \sin(2\pi t)$ will strongly attract solution $\Rightarrow$ stiff problem

$\dot z = \begin{bmatrix} -\lambda z_1 + \beta \sin(2\pi z_2) \\ 1 \end{bmatrix}$ $\quad z(t) = \begin{bmatrix} q(t) \\ t \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$

for gauss. coll. SSM: $\lim_{|z| \to \infty} S(z) = 1$ $\quad |h\lambda| \gg 1 \Rightarrow |S(\lambda h)| \approx 1$: $q_k = (S(z))^k q_0$

$\Rightarrow$ very slow decay of the ssm solution

very fast decay of $t \to q(t)$

We want $S(\infty) = 0$ (satisfied for implicit euler)

(desirable for stiff problems) → GC-SSM $|S(\infty)| = 1$ (gauss-collocation)

L-stability

RK is L-stable (=asymptotically stable) if its stability function satisfies

i) $\mathrm{Re}(z) < 0 \Rightarrow |S(z)| < 1$ (implied by A-stability)

ii) $\lim_{\mathrm{Re} z \to -\infty} S(z) = 0$

L-stability = A-stability + $S(\infty) = 0$

construction of L-stable methods

$S(z) = 1 + zb^T (I - Az)^{-1} \mathbb{1} = 1 + b^T \left(\frac{1}{z} I - A\right)^{-1} \Rightarrow S(0) = 1 - b^T A^{-1} \mathbb{1} \overset{!}{=} 0$

$b = A_{j,:}$ (j^{th} row) $\Rightarrow b^T A^{-1} = e_j^T \Rightarrow b^T A^{-1} \mathbb{1} = 1$

→ $S(\infty) = 0$ $\forall$ RK-SSM with Butcher scheme $\begin{array}{c|ccc} c_1 & a_{11} & \dots & a_{1s} \\ \vdots & \vdots & & \vdots \\ c_s & a_{s,1} & \dots & a_{s,s} \\ \hline & b_1 & \dots & b_s \end{array}$

s-stage **Gauss-Radau collocation SSM** of order $2s-1$, L-stable

$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array}$

$\begin{array}{c|cc} 1/3 & 5/12 & -1/12 \\ 1 & 3/4 & 1/4 \\ \hline & 3/4 & 1/4 \end{array}$ order 3

$\begin{array}{c|ccc} \frac{4-\sqrt{6}}{10} & & & \\ \frac{4+\sqrt{6}}{10} & " & " & " \\ 1 & " & " & " \\ \hline & \frac{16-\sqrt{6}}{36} & \frac{16+\sqrt{6}}{36} & \frac{1}{9} \end{array}$ order 5

SSM: IMPLICIT RKSSM

idea: fixed small number of Newton steps to compute increments

⇔ linearize increment equation

ex: implicit Euler, $\dot q = f(q)$ $\quad q_1 = q_0 + hf(q_1)$

linearization around q_0: $q_1 = q_0 + h\left(f(q_0) + Df(q_0)(q_1 - q_0)\right) \doteq q_0 + \underbrace{(I - hDf(q_0))^{-1}}_{\text{invertible for sufficient small } h} hf(q_0)$

(semi implicit euler SSM)

$\sigma(I - hDf(q_0)) = 1 - h\sigma(Df(q_0))$

semi implicit midpoint method

linearization: $q_1 = q_0 + hf\left(\frac{1}{2}(q_0 + q_1)\right) \doteq q_0 + h\left(f(q_0) + \frac{1}{2}Df(q_0)(q_1 - q_0)\right)$

$\left(1 - \frac{1}{2}hDf(q_0)\right)(q_1 - q_0) = hf(q_0)$

→ Is one newton step enough for all implicit RKSSM

good news: fixed # of newton steps is enough

ROW (Rosenbrock-Wanner) semi-implicit RK METHODS

$(I - ha_{ij}J)k_i = f\left(q_0 + h\sum_{j=1}^{i-1}(a_{ij}+d_{ij})k_j\right) - h\sum_{j=1}^{i-1} d_{ij}k_j$ $\quad J = Df(q_0)$

$q_1 = q_0 + \sum_{j=1}^s b_j k_j$

practice: determine a_{ij}, d_{ij}, b_j from order conditions

→ see paper

how to explore L-stability of ROW method?

i) Apply SSM to $\dot q = \lambda q$ $\quad (f(q) = \lambda q)$

ii) we get $q_1 = S(h\lambda) q_0$ → more general $Q(h\lambda) q_1 = P(h\lambda) q_0$, PQ polynomials

iii) Verify $\lim_{z \to \infty} S(z) = 0$ (L-stab) $\quad \mathrm{Re}(z) < 0 \Rightarrow |S(z)| < 1$ (A-stability)

Use Theorem: if S rational, defined on $\mathbb{C}^- \cup i\mathbb{R}$, $\lim_{z \to \infty} S(z) = 0$

$\Rightarrow |S(z)| < \sup_{\tau \in \mathbb{R}} |S(i\tau)| \quad \forall z \in \mathbb{C}^-$

Remark: $\dot q = \lambda q$ $\quad q(h) = e^{h\lambda} q(0)$ (ideal stability function)

$\leftrightarrow q_1 = S(h\lambda) q_0$

for SSM: $S(z) \overset{!}{=} e^z$ for $|z| \ll 1$ $\quad$ SSM of order $p \Rightarrow S(z) - e^z = O(|z|^{p+1})$ for $z \to 0$

adaptive integrator for stiff problems in MATLAB

ex ode23s : 2,3 = different order methods for stepsize control error

↳ stiff IVP

Template:

```
#include <Eigen/Dense>
#include <iostream>
using namespace Eigen;
using namespace std;
int main() { ... , return 0 }
```

```
#include cmath
setw(15)
cout.precision(7);
```

```
template<class Matrix>
void f(const Matrix &d, ...)
  // decompose vectors, whatever
A << ..., ..., ... // row major
using Vector = Eigen::VectorXd;
```

reshape

```
auto tmp = R.template triangularView<Eigen::Upper>();
x = A.lu().solve(b); // returns x = A^-1 b  (solves Ax = b)
```

function pointer

```
double f(double x){ return sin(x); }
void Newton( double (*f)(double), ... ){ ... }
```

MATLAB: quad(@(t) a*sin(t).*sinh(t), -1, 1, eps)

```
// call:
Newton(&f, ...)
```

limits

```
#include <limits.h>
double eps = numeric_limits<double>::epsilon(); // 2,2204e-16
```

Lambda functions

```
auto f = [&](double x){ return sin(x); };
[auto = std::function<double(double)>]
```

flags: -std=c++11

cmake Find Eigen3.cmake

```
cmake .        // initialize
make           // compile all        make problemX  // compile only this
./problemX     // execute
```

```
g++ -std=c++11 -I/usr/include/eigen3 -Wno-deprecated-declarations
```

```
Matrix<typename scalar, int RowsAtCompileTime, int
  ColsAtCompileTime>
#include <Eigen/Dense>
template<typename Scalar>
void eigenType Demo (unsigned int dim){
  using dynMat_t = Eigen::Matrix<Scalar,Eigen::Dynamic, Eigen::Dynamic>;
  using dynColVec_t =                                  ,1,
  using dynRowVec_t =                                  ,1,Eigen::Dynamic>;
  using index_t = typename dynMat_t::Index;
  using entry_t =                    ::Scalar;
  dynColVec_t colVec(dim);
  dynRowVec_t rowVec(dim);
  for (index_t i= 0; i< colVec.size(),++i ) colVec(i)= (Scalar) i;
                                  rowVec                rowVec(i)= (Scalar) 1/(i+1);
  dynMat_t vecprod = colVec * rowVec;
  const int nrows = vecprod.rows();
            ncols =         .cols();
```

```
//dest allocate space, no init
Eigen::MatrixXd A(rows,cols);
               B = MatrixXd::Zero(rows,cols);
                           Ones
                           Constant(rows,cols,value)
                           Random(rows,cols);          //[0,1]
                           Identity
"size of A=" << A.rows() << "x" << A.cols() << endl;
```

```
template<typename MatType> void blockAccess(Eigen::MatrixBase
    <MatType>& M){
using index_t = typename Eigen::MatrixBase<MatType>::Index;
      entry_t =                                    ::Scalar;
const index_t nrows(M.rows());
              ncols =      .cols();
cout << "Matrix M=" << endl << M << endl;
index_t p= nrows/2, q= ncols/2;
for(index_t i=0; i< min(p,q); i++)
    cout << "block" << i << " , " << i << " , " << p << " , " << q << " = " << M.block(i,i,p,q);
M.block(1,1,p,q) += MatrixXd::Constant(p,q, 1.0);
MatrixXd B(M.block(1,1,p,q));
```

```
//special sub matrices          M.topRows(p)
                                M.bottomRows(p)
                                M.leftCols(q)
                                M.rightCols(q)
```

```
//upper triangular
const MatrixXd T = M.template triangularView<Upper>();
                                          ↖lowers()      *= -1.5 ;
Matrix<double, Dynamic, Dynamic, ColMajor> mCM(nrows,ncols);
                                    RowMajor> mRM =
```

```
function y = expeval (x,tol)                    | summation of
y=1; term=1;   k=1;          %init              | exponential series
while
   (abs(term) > tol*min(y,1))
    term = term *x/k;        % next summand
    y = y+ term;             % summation
    k = k+1;
end
```

```
MatrixXd mat = MatrixXd::Identity(n,n) + VectorXd::Constant(n,1.0)*RowVectorXd::
                                          constant(n, 10)
//untested invertible matrix
MatrixXd B = MatrixXd::Random(n,2);  //rhs
//solve: linear system using various decompositions

MatrixXd X= mat.lu().solve(B);
          X2      .fullPivLU().solve(B)
                  .householderQr().solve(B);
                  .llt().solve(B);          //cholesky
                  .ldlt().solve(B);         //robust "

cout << "|X-X2|" << (X-X2).norm() << endl;
```

```
void mod_newt_syst() {
  // using Vector = Eigen::VectorXd;
MatrixXd A= MatrixXd::Random(4,4);
VectorXd c= VectorXd::Random(4);
A = (A*A.transpose()).eval();   //A is now symmetric
c = 0.cwiseAbs;                 // c>0 uniformly

auto f= [&A,&c](const VectorXd &x){ VectorXd tmp= A*x + c.cwiseProduct(
          x.array().exp().matrix()).eval(); return tmp;};

function<MatrixXd(const VectorXd &)> df=[&A,&c](const VectorXd &x){
   MatrixXd C=A;
   VectorXd temp= c.cwiseProduct(x.array().exp().matrix());
   C+= temp.asDiagonal(); return C;};

vector<double> err;
//define button for breaking condition
auto ferr = [&err] (VectorXd &x) { if(err.size()>0) return err.back();
                                   else return 1.;};
VectorXd x_system= VectorXd::Zero(c.size());   //initial guess

auto newt_system_step = [&f, &df, &err](const VectorXd &x,VectorXd &x_new){
    mod_newt_step_system(x,x_new,f,df); double e=(x-x_new)
        .norm()/x_new.norm(); err.push_back(e); };
general_nonlinear_solver (newt_system_step,x_system,ferr);
//sanity check
cout << x_system  <<  F(x_system);
}
```

```
double polyfit(const VectorXd &t, const VectorXd &y, const double &x){
  int n= t.size()    // assert size matches
  VectorXd L(n); //set lagrange polynomials
  for(int i=0;i<n;i++){
    L(i)=1;
    for(int j=0;j<n;j++){ if(j!=i){
      L(i)*= (x-t(j)) / (t(i)-t(j));
    }
  }
  double ans=0;  //compute p(x)
  for(int i=0;i<n;i++){
    ans += y(i)*L(i);
  }
  return ans;
}
```

```
double aitkenNev(const VectorXd &t, const VectorXd &y, const double &x){
int n= t.size();   //assert size matches
int k,l;
double p[n][n];
for(k=0;k<n;k++) p[k][k]= y(k);  //initialize
for(l=1;l<n;l++){
  for(k=l-1; k>=0;k--){   //compute recursively
    p[k][l]= ( (x-t(k))*p[k+1][l] - (x-t(l))*p[k][l-1] ) / (t(l)-t(k));
}}return p[0][n-1];
}
```

```
double Pnx(double x,int n){ //evaluate Legendre polynomials in x upto
                            degree n
  VectorXd Px(n+1);
  Px(0)=1; Px(1)= x;
  for(int k=2; k<=n;k++){
    Px(k)= (2*k-1.)/k*x*Px(k-1) - (k-1)/k*Px(k-2);
  }return Px(n);
}
```

```
MatrixXd gaussPts(int n, double dal=1e-10 double atol= 1e-12){
                  //find gauss points using secant method (+ regula falsi)
  MatrixXd zeros(n,n);
  double x0, x1, f0, f1, s;
  for(int k=1; k<=n;k++){
    for(int j=0;j<k+1;j++){
      if(j==1) x0 = -1;
      else     x0 = zeros(j-2,k-2);  //initialize initial guesses
      if(j==k) x1 = 1.;
      else     x1 = zeros(j-1,k-2);

    //secant method
    f0 = Pnx(x0,k);
    for(int i=0; i<1e4; i++){
      f1 = Pnx(x1,k);
      s= f1*(x1-x0) / (f1-f0);
      if(Pnx(x1-s,k)*f1<0){  x0=x1; f0=f1; }
      x1 = x1-s;
      if( (abs(s)<max(atol, dtol*min(abs(x0),abs(x1)))) ){
          zeros(j-1,k-1) = x1;
          break;
      }
    }}
  }
  return zeros;
}
```

rk3prog.cpp

```
#include "rkIntegrator.hpp"
int main(void){
  unsigned int s=3;
  MatrixXd A(s,s); A<< ...   // Butcher scheme
  VectorXd b(s); b<<
//coefficients and nodule for function
double w1=3; double β1=0.1,
double w2=2; double β2=0.1,
auto f= [&w1,&w2,&β1,&β2](const VectorXd &y){
   auto temp = y;          // so a workstate copy
   t(0) *= w1 - β1*y(1);       t(1)*= -w2+ β2*y(0);
   return temp;
  };
unsigned int d=2;          //dimension of state space
double T=10.;              //final time for model
VectorXd y0(d); y0<< 100.,5.; // initial value of model
vector<unsigned int> N= {128, 256, ... , 16384};
VectorXd yex(d); yexact 0,2e-  , 2.37...  ;  // exact values

//initialize RK with butcher scheme
RKIntegrator<VectorXd> RK(A,b);

double errold;
for(unsigned int i=0; i<N.size(); i++){
    auto res= RK.solve (f,T, y0, N[i]);
    double er = (res.back()-yex).norm();          // er out << log2(errold/er),
    errold =er;
}
```

general_nonlinear_solve.h

```
#pragma once
template<class stp_funct, class Vector, class err_funct>

bool general_nonlinear_solver(stp_funct &&step, Vector &x, err_funct &errf,
                              double eps= 1e-8, int max_itr=100){

  auto new= x; // temp
  for(int itr=0;;){          //forever until break
    double r= errf(x);  //compute residual
    cout << "step" << itr << ": error: " << r << endl;
    step(x,x_new);  //advance to new step
    //termination condition
    if( r<eps){ cout << "converged in" << itr << "with error" << r << "<" << eps;
                return true;
    }
    if(++itr >= max_itr){ cout<<"not converged"<<-- max_itr <<"reached"
                return false;
    }
    x= xnew;
  }
}
```

modnewt.cpp

```
#include "general_nonlinear_solver.h"

template<typename argument, class function, class jacobian>
void mod_newt_scalar(const argument &x, argument &x_next,
                     function &&f, jacobian &&df){
  argument y= x+f(x)/df(x);          //particular case
  x_next = y- f(y)/df(x);
}
template<typename                                  > //same
void mod_newt_step_system(const argument &x                    ){ //same
   auto lu = df(x).lu();          //decompose lu in the latter
   argument y= x + lu.solve(f(x));
   x_next   = y - lu.solve(f(y));
}
void mod_newt_ex0(){          //solve scalar nonlinear eq. system
  const double a= 0.1231;
  auto f=[&a](double x){ return atan(x)- a; };
  auto df=[](double x) { return 1./(x*x+1.); };
  const double x_ex= tan(a);          //exact sol
  vector<double> sol; vector<double> err;

  auto errf = [&err, x_ex](double &x){double e=abs(x-x_ex); err.push_back(e);
                                      return e;}
  double x_scalar = 5;
  sol.push_back(x_scalar); errf(x_scalar);

  auto newt_scalar_step= [&sol,&df](double x, double &x_new){ //double performs
      mod_newt_step_scalar(x,x_new,f,df);          step
      sol.push_back(x_new); };
  general_nonlinear_solver (newt_scalar_step, x_scalar, errf); //perform solution
  cout << x_scalar << endl;                                    //print sol

  cout <<      sol                err                order
  for(i:0;i<sol.size();++i){  cout << sol.at(i)    << err.at(i);
  if i>=3) { double order= (log(err.at(i)) - log(err.at(i-1)))/(log(err.at(i-1)) -
      cout<<order;                                  log(err.at(i-2)))
  }
}
```

```
void solvebwelsh(int n, VectorXd &w, VectorXd &x){
  w.resize(n);
  x.resize(n);                          //for gauss-legendre
  if(n==0) { w(0)= 2; x(0)=0; }
  else{
    VectorXd t(n-1);
    MatrixXd J = MatrixXd::Zero(n,n);
    for(int i=1; i<n;++i){
      double d= i)/sqrt(4*i*i -1);
      J(i,i-1)=d;
      J(i-1,i)=d;
    }
    EigenSolver<MatrixXd> eig(J);
    x= eig.eigenvalues().real();          | w = eig.eigenvectors().real()
    w= 2*w.topRows<1>().cwiseProduct(w.topRows<1>());
  }
}
```

```
double quad(funct &&f, const VectorXd &w,const VectorXd &x,double a,double
    double I=0;
    for(int i=0; i<w.size(); ++i){
      I += f( (x(i)+1)*(b-a)/2 + a) * w(i);
    }
    return I*(b-a)/2;
}
double quadInf(int n, funct&&f){
  VectorXd w,x;
  solvebwelsh(n,w,x);
  auto ftilde = [&f](double x){return f(cos(x)/sin(x)) /pow(sin(x),2);};
  return quad(ftilde,w,x,0,M_PI);
}
```

.hpp

```
template<class State>
class RKIntegrator {
public: RKIntegrator(const MatrixXd &A, const VectorXd &b): A(A),b(b),s(b.si
    assert( A.cols()== A.rows());  //can add error msg
    assert(A.cols()== b.size());
  }
  template<class Functions>
  vector<State> solve(const Function &f, double T, const State &y0, unsigned
    int N) const {
    double h= T/N; // init step size
    vector<State> res; res.reserve(N);  //will contain all steps
    res.push_back(y0);
    State ytemp1= y0;
    State ytemp2= y0;
    State* yold = &ytemp1; // pointers to swap previous values
    State* ynew= &ytemp2;

    for(unsigned int k=0; k<N; ++k){ //compute, save & swap next
        step(f,h, *yold, *ynew);
        res.push_back(*ynew);
        swap(yold, ynew);          //it's standard
    } return res;
  }
private: template<class Function> // perform single step for one
                  void step(const Function &f, double h, const State &y0, State &
    const {
    y1= y0;  // hold next value
    vector<State> k; k.reserve(s); // reserve space for increments
    for(unsigned int i=0; i<s;++i) {
      State incr = y0;
      for(unsigned int j=0; j<i;++j) {
        incr += h*A(i,j)*k.at(j);
      }
      k.push_back(f(incr));
    } y1 += h* b(i) * k.back();
  }
  const MatrixXd A;
  const VectorXd b;
  unsigned int s;
};
```

Aitken-Neville

```
double pLagrange(int k){
  assert(k>=1 && k<5 && "k must be... ");
  vector<double> t= {-, -, -, ... };
  vector<double> U= {-, -, -, ... };
  for(int l=0, l<k-1; ++l) {                  //loop over each degree (|x[left->right]|)
    for(int i=0; i<k-l-1; ++i) {              //loop over each node (loop->below)
      //store values in U (overwrite all)
      // has to be adjust by 1
      U[i]= (-t[i]*U[i+1] + t[i+l]*U[i]) /(t[i+l]-t[i]);
    }
  }
  return U[0]; //contains value of p
}
```

NATCSI

```
                //natural cubic spline interpolant
class NatCSI {
public: NatCSI(const vector<double> &t, const vector<double> &y):
          t(t), y(y), h(t.size()-1), c(t.size()){
    assert(t.size()== y.size())
    n= t.size();
    for(int i=0; i<n-1; ++i){ //vector containing increments
      h(i) = t[i+1] - t[i];
      assert(h(i)>0)          // check t is sorted
    }
    SparseMatrix<double> A(n,n);  // system matrix and RHS
    VectorXd b(n);
    A.reserve(3);  //sparse reserve space
    A.coeffRef(0,0) = 2/h(0);          | (n-1,n-2) = 1/h(n-2) // fill in
               (0,1) = 1/h(0);         | (n-1,n-1) = 2/h(n-2)  natural
                                                               conditions
    double bold= (y[1]-y[0])/(h(0)*h(0)); //reuse computation)
    b(0)=3*bold;
    for(int i=1; i<n-1; ++i){
      double hinv = 1/h(i);          //precompute b;
      A.coeffRef(i,i-1) = hinv;
                (i,i) = 2./h(i-1) + 2./h(i);   //fill in A
                (i,i+1) = hinv;
      double bnew = (y[i+1]-y[i])/(h(i)*h(i));
      b(i) = 3*(bnew+bold);
      bold=bnew;
    } b(n-1)= 3*bold;  //fill in natural conditions
      A.makeCompressed();      cout << A;
      SparseLU<SparseMatrix<double>> lu; // factorize A and solve Ac=b
      lu.compute(A);
    c= lu.solve(b);
  }
  double operator()(double x) const {
    assert(t[0]<= x && x<= t[n-1]) //assert x in range
    auto j= (lower_bound(t.begin(), t.end(), x) - t.begin()) -1;
    if(j==-1)++j;                  //find j s.t. x∈ [t_j, t_j+1]
    //precompute tau and evaluate spline
    double tau= (x-t[j])/h(j); double tau2= tau*tau; double tau3= tau2*
    return   y[j] *          (1. - 3*tau2 + 2*tau3) +
             y[j+1]*         (3*tau2 - 2*tau3) +
             h(j)*c(j)*      (tau - 2*tau2 + tau3) +
             h(j)*c(j+1)*    (-tau2 + tau3);
  }
private: vector<double> t,y;   //nodes & values to compute, splines
         VectorXd h,c;         // intervals and coefficients of spline
         int n;                // size
```

```
#include <stdlib>
        <vector>
        <cmath>
        <iomanip>
```